A MARSHALL CAVENDISH **42** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol. 4 **No 42**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front Cover, Dave King. Page 257, Nick Farmer. Page 258, K. Goebel/ZEFA/Hussein Hussein. Pages 260, 261, Ray Duns. Page 262, Tony Stone/Hussein Hussein. Pages 264, 266, 267, Alan Baker. Pages 269, 270, 272, Dave King. Pages 277, 279, Nick Farmer/Nick Mijnheer. Page 283, Digital Arts. Pages 284, 286, 288, Peter Bentley.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd,
Department 980, Newtown Road,
Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services,
Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd,
Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

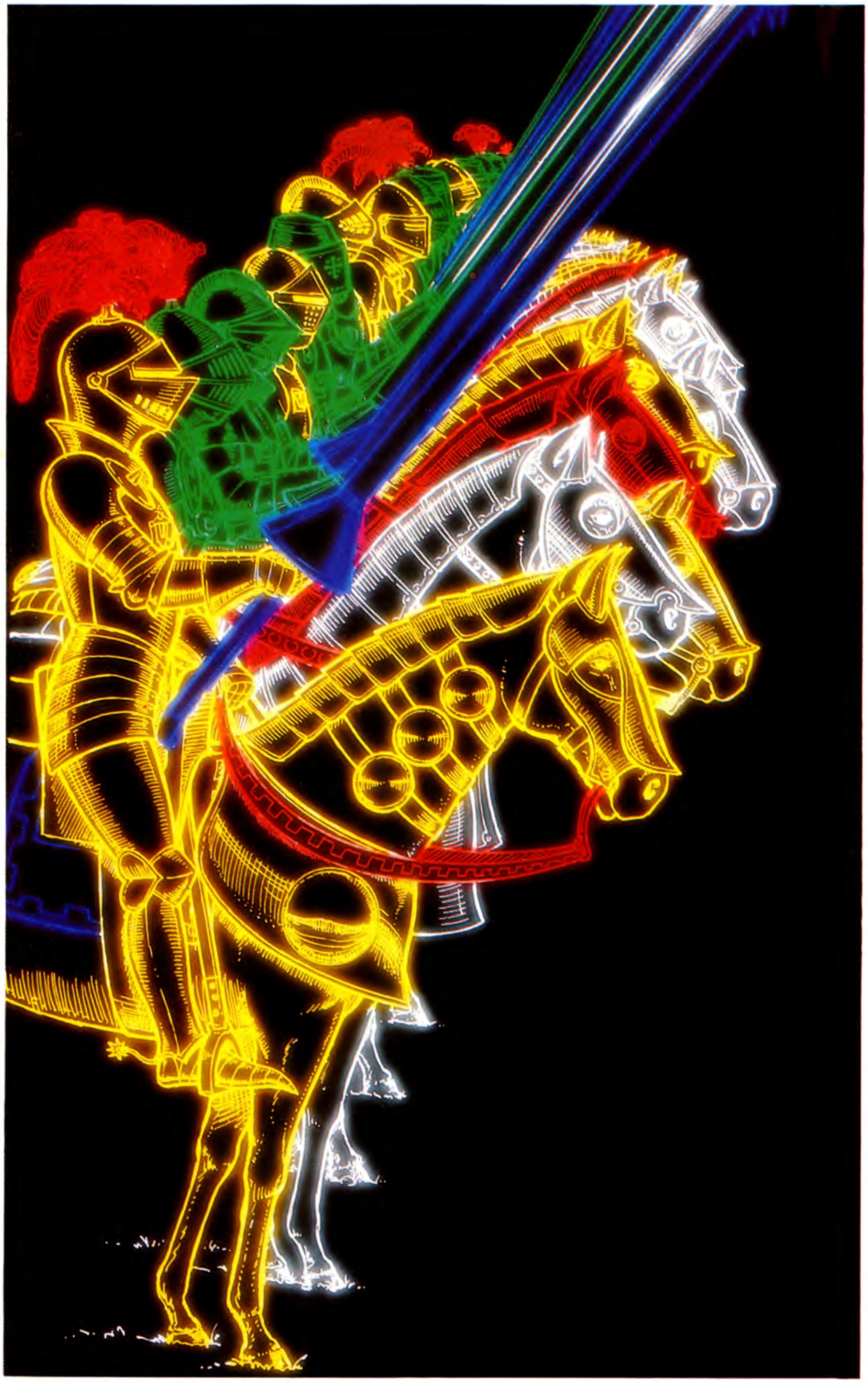
The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

WARGAMING: THE ART OF COMMAND

- INITIALIZATION
- GIVING ORDERS
- MOVING AND DIRECTIONS
- STATUS
- THE COMPUTER'S MOVE



Instil some discipline into the unruly rabble and stop them milling about the battlefield. Issue orders with these new routines and put your masterplan into action

Now you have the routines which will place your armies on the battlefield and you have added the movement routine, you will want to be able to issue commands to your army.

There are a number of factors connected with your troops which will affect the way they perform on the battlefield:

- Factor one, the unit's current order (what you told them to do last time)
- Factor two, the direction of current movement
- Factor three, weaponry
- Factor four, armour
- Factor five, initial strength
- Factor six, current strength
- Factor seven, morale or attitude
- Factor eight, position
- Factor nine, terrain

These factors correspond to the nine elements in the troop array that you set up last time. Position and terrain have already been filled, in connection with the movement routine, and in this article you will fill in the other elements.

At the start of the game initial values will have to be fed into the troop array. Weaponry and armour values will be fixed. Morale will be roughly the same each time, but may well vary slightly—peasants are probably not going to be very keen on fighting, whereas knights, having to keep up appearances, are never cowardly. Strength could vary quite considerably from game to game. Orders and direction could be set to arbitrary values at the start, but in Cavendish Field, everyone starts from Halt, meaning that direction is unimportant.

The following routines set up the troop array either by reading in appropriate data, or performing a calculation, or both. In fact, once you have the program working at the end of part four, you may wish to try experimenting with different ways of setting up these initial values.

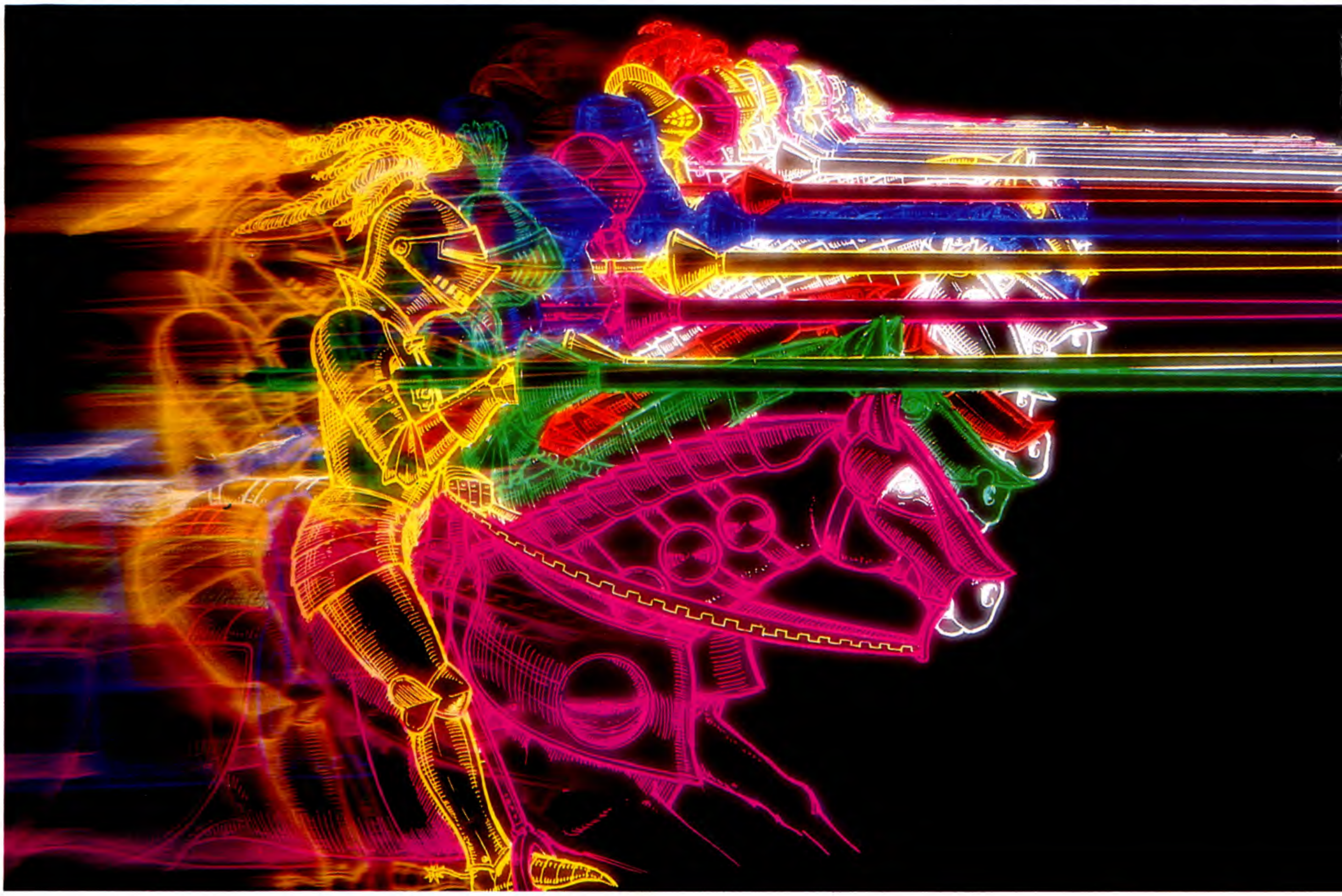
VISITING THE QUARTERMASTER

This routine initializes the unfilled array elements:

```
190 REM Initialization
200 LET vc=Ø: LET de=Ø
310 REM
320 PAPER 7
330 INK Ø
340 CLS
360 DIM t$(8,12): DIM o$(5,12): DIM
    w$(5,9): DIM m$(5,12): DIM a$(4,12):
    DIMr$(4,12):DIMc(8)
380 FOR i=1 TO 5: READ o$(i),
    w$(i),m$(i): NEXT i
390 FOR i=1 TO 8: READ t$(i): NEXT i
400 FOR i=1 TO 4: READ a$(i),
    r$(i): NEXT i
410 LET u$=CHR$148+CHR$149+CHR$
    150+CHR$150+CHR$151+CHR$151+
    CHR$152+CHR$152:LET u$=u$+u$
415 LET x$="NnYy"
420 RETURN
2760 DATA "fire","none","cowardly","halt",
    "bow","unwilling"
2770 DATA "move","sword","willing",
    "status","axe","brave"
2780 DATA "rout","lance","valiant"
2790 DATA "knights","sergeants","men-at-
    arms","men-at-arms","archers","archers",
    "peasants","peasants"
2800 DATA "none","plains","jerkin",
    "village","chainmail","woods","plate",
    "hills"
2810 DATA 5,4,3,5,3,3,4,3,2,3,3,1,2,2,1,2,3,2,
    3,2,Ø,3,1,Ø
```

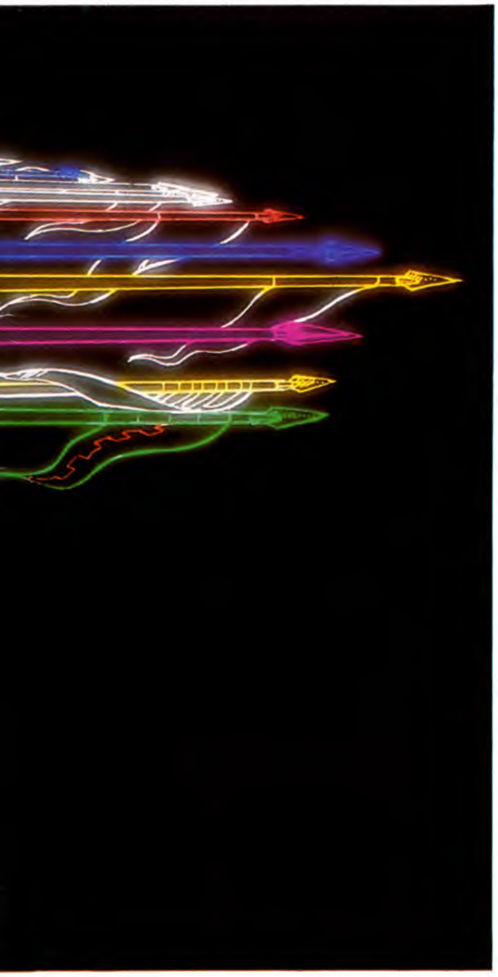
```
190 REM INITIALIZATION
195 DEF FNR(X)=INT(RND(1)*X)+1
200 VC=Ø:DE=Ø
210 POKE53281,5
220 PRINT CHR$(144)
340 PRINT "♥"
360 DIM T$(7),O$(4),W$(4),M$(4),A$(3),
    R$(3)
380 FOR I=Ø TO4:READ O$(I),W$(I),M$(I):
    NEXT I
390 FOR I=Ø TO7:READ T$(I):NEXT I
400 FOR I=Ø TO3:READ A$(I),R$(I):NEXT I
415 W$="NNYY"
420 RETURN
2870 DATA FIRE,NONE,COWARDLY,HALT,
    BOW,UNWILLING
2880 DATA MOVE,SWORD,WILLING,STATUS,
    AXE,BRAVE
2890 DATA ROUT,LANCE,VALIANT
2900 DATA KNIGHTS,SERGEANTS,MEN-AT-
    ARMS,MEN-AT-ARMS,ARCHERS,ARCHERS
2910 DATA PEASANTS,PEASANTS
2920 DATA NONE,PLAINS,JERKIN,VILLAGE,
    CHAINMAIL,WOODS,PLATE,HILLS
2930 DATA 4,3,3,4,2,3,3,2,2,2,2,1,1,1,1,1,2,2,
    2,1,Ø,2,Ø,Ø
```

```
190 DEF PROCinit
200 vic=FALSE:def=FALSE
360 DIM type$(7),order$(4),wpn$(4),mor$(4),
    arm$(3),ter$(3)
380 FOR i=ØTO4:READ order$(i),wpn$(i),
    mor$(i):NEXT
390 FORi=ØTO7:READ type$(i):NEXT
400 FORi=ØTO3:READ arm$(i),ter$(i):NEXT
420 ENDPROC
2570 DATA fire,none,cowardly,halt,bow,
    unwilling,move,sword,willing,status,axe,
    brave,rout,lance,valiant,knights,sergeants,
    men-at-arms,men-at-arms,archers,archers,
```



```
    peasants,peasants,none,plains,jerkin,village,
    chainmail,woods,plate,hills
2620 DATA 4,3,3,4,2,3,3,2,2,2,2,1,1,1,1,1,2,2,
    2,1,0,2,0,0
```

```
310 REM
330 COLOR 4,1
340 PCLS
360 DIM T$(8),O$(5),W$(5),M$(5),A$(4),
    R$(4)
380 FOR J=1 TO 5:READ O$(J),W$(J),
    M$(J):NEXT J
390 FOR J=1 TO 8:READ T$(J):NEXT J
400 FOR J=1 TO 4:READ A$(J),R$(J): NEXT
415 X$="NnYy"
420 RETURN
2760 DATA FIRE,NONE,COWARDLY,HALT,
    BOW,UNWILLING
2770 DATA MOVE,SWORD,WILLING,STATUS,
    AXE,BRAVE
2780 DATA ROUT,LANCE,VALIANT
2790 DATA KNIGHTS,SERGEANTS,MEN AT
    ARMS,MEN AT ARMS,ARCHERS,
    ARCHERS,PEASANTS,PEASANTS
2800 DATA NONE,PLAINS,JERKIN,VILLAGE,
    CHAINMAIL,WOODS,PLATE,HILLS
```



```
2810 DATA 5,4,3,5,3,3,4,3,2,3,3,1,2,2,1,2,3,2,
    3,2,0,3,1,0
```

Initial values are READ from the DATA lines, along with words which are equivalent to the various numeric values fed into the array. The words will be used in the text window to display the status of the battle clearly.

GIVING ORDERS

The whole game hinges on the player giving orders to the army—without this there would be no combat, and no winning or losing.

There are four orders you can give to your troops: Fire, Halt, Move and Status. The order to fire only applies to the archers, and they will fire on request even if there is no target around.

Giving orders to units has its own trade-offs. The more realistic the game, the more complex the giving of orders. The simpler the game, the easier it is to give orders. One result of this is that a realistic game will be very much more difficult to play; and if you wish to have a game that's easy to play, you'll have to make sacrifices in terms of nearness to reality. After you have written some wargames you'll probably formulate your own ideas on the best compromise.

The routine in Cavendish Field goes through each of the player's units in turn, identifying the unit to which orders can be given by changing its colour on the display. It also prints a message which asks if the standing orders for that unit (number x) are to be changed—the units continue doing whatever they were instructed by the last order issued to them until the player changes the order. If the player doesn't want to change the orders to that unit, all that needs to be done is to press the N key. If Y is pressed, then a menu of the four possible commands is given.

If Fire is selected, and the unit isn't archers, then another order is requested. Halt is self-explanatory. Move asks for a direction to be input. No check is made at this stage if a move is possible, so you may wish to modify the game so that no move option is displayed if no move is possible—but remember, adding input checks slows the game down. Status responds with a description of the current state of the unit.

SELECTING A UNIT

When it is the player's turn, the first unit is highlighted, followed by the other seven in turn. The unit which changes colour is the unit which the player must consider—should the orders be changed? As the unit is highlighted, prompts appear in the text window, along with the current orders.

```
1380 REM Select unit for orders
1390 GOSUB 2540
1400 INK 0
1410 PRINT FLASH1;AT T(i,8),T(i,9);u$(i)
1420 PRINT AT 17,0;"Unit number□";i;"□";
    t$(i);"□□□□□□□"
1430 PRINT AT 18,0;"Current orders are
    to ";o$(T(i,1));"□";
1440 IF T(i,1)=3 THEN PRINT AT
    18,28;i$(T(i,2))
1450 REM Dummy for repeat loop
1460 PRINT AT 19,0;"Do you want to change
    orders (Y/N)?"
1465 LET y=0: LET y$=INKEY$
1466 IF y$="" THEN GOTO 1465
1470 FOR k=1 TO 4
1475 IF x$(k)=y$ THEN LET y=k
1480 NEXT k
1490 IF y=0 THEN GOTO 1450
1500 RETURN
```

```
1380 REM SELECT UNIT FOR ORDERS
1390 GOSUB 2540
1400 CL=6
1410 P=T(I,7):Q=T(I,8):GH=VAL(MID$
    (U$,I+1,1))+67:GOSUB2600
1420 PRINT "UNIT NUMBER□";I+1;"□";
    T$(I)
1430 PRINT "CURRENT ORDERS:□";O$
    (T(I,0));"□";
1440 IF T(I,0)=2 THEN PRINT
    MID$(I$,T(I,1),1)
1445 PRINT"□"
1450 PRINT"DO YOU WANT TO CHANGE
    ORDERS? (Y/N)"
1460 REM DUMMY FOR LOOP
1465 Y$="":GETY$
1470 FOR K=1 TO4
1475 IF MID$(W$,K,1)=Y$ THEN Y=K
1480 NEXT K
1490 IF Y=0 THEN 1460
1500 RETURN
```

```
1380 DEF PROCunitsel
1390 PROCclean
1400 COLOUR 0
1410 PRINT TAB(T%(i,7)+1,T%(i,8)+1);
    MID$(unst$,i+1,1)
1420 PRINT TAB(2,23);"Unit number□";i+1;
    "□";type$(i);"□□□□□□□"
1430 PRINT TAB(2,24);"Current orders:□";
    order$(T%(i,0));"□";
1440 IF T%(i,0)=2 THEN PRINT
    MID$(dir$,T%(i,1),1)
1450 REPEAT
1460 PRINT TAB(2,25);"Change orders (Y/N)?"
1470 yn$=GET$
```

```
1480 yn=INSTR("NnYy",yn$)
1490 UNTIL yn
1500 ENDPROC
```

```
1380 REM SELECT UNIT FOR ORDERS
1390 GOSUB 254Ø
1400 COLOR 4
1410 DRAW"BM"+STR$(T(I,9)*8)+","+
    STR$(T(I,8)*8):UU=VAL(MID$(U$,I,1)):
    A$=UC$(UU):GOSUB 3Ø3Ø
1420 DRAW"BMØ,144":A$="UNIT "+
    STR$(I)+"□"+T$(I)+"□□□□□
    □□":GOSUB 319Ø
1430 DRAW"BMØ,152":A$="ORDERS ARE
    TO□□□"+O$(T(I,1))+"□":
    GOSUB 319Ø
1440 IF T(I,1)=3 THEN DRAW"BM16Ø,152":
    A$=MID$(I$,T(I,2),1):GOSUB 319Ø
1450 REM
1460 DRAW"BMØ,16Ø":A$="CHANGE
    ORDERS□□Y OR N":GOSUB 319Ø
1465 Y=Ø:Y$=INKEY$
1466 IF Y$="" THEN 1465
1470 Y=INSTR(1,X$,Y$)
1490 IF Y=Ø THEN 145Ø
1500 RETURN
3000 X9=PEEK(2ØØ):Y9=PEEK(2Ø2):LINE
    (X9,Y9)-(X9+7,Y9+7),PRESET,BF
3010 POKE 2ØØ,X9:POKE 2Ø2,Y9:DRAW A$
3020 RETURN
3030 X9=PEEK(2ØØ):Y9=PEEK(2Ø2):
    C9=PEEK(178):COLOR 2:LINE(X9,Y9)-
    (X9+7,Y9+7),PSET,BF
3040 POKE 178,C9:GOTO 3Ø1Ø
3050 REM DATA FOR LETTERS & DIGITS
3060 DATA BR4,BDD5RU3NR2U2ERFND5BR4
    BU,D6R3EUHEUHBR5,NR5D6R5BR3BU6,
    NR3D6R3EU4BUBR4,NR5D3NR3D3R5BR3
    BU6,NR5D3NR3D3BR8BU6,NR5D6R5U2
    BU4BR3
3070 DATA ND6D3R5D3U6BR3,R5L2D6L3R5
    BR3BU6,BD5RFREU5L2BR6,ND6D3R3FD2
    BU4U2BR4,D6R5BR3BU6,ND6DR5ND5
    UBR3,ND6DR2D3R3D2U6BR3
3080 DATA D6R5U6NL5BR3,D6U3R5U3NL5
    BR3,D6UR3FRBU2U4NL5BR3,D6U2R3FDU
    BU2NL2U3NL4BR4,NR5D3R5D3NL5BU6
    BR3,R5L2ND6BR5,D6R5U6BR3
3090 DATA D4RFDRUEU4BR4,D6UR5DU6
    BR3,D2RFGND2ERFND2HEU2BR4,D2RFD3
    RU3EU2BR4,R4D2GLGD2R4BU6BR3
3100 DATA BR3LGD4FREU4BR4BU,BR2DNL2
    D5L2R5BU6BR3,BDRERFDGL2D3R4BU6
    BR3,BDRERFDGFDGLHLBU5BR8,D4R5UD3
    BU6BR3,NR5D2R3FD2GLHLBU5BR8,BDB
    R5LHLGD2NR2D2FREUBU4BR4
3110 DATA R5D2LGLGD2BR7BU6,BR3LGDF
    GDFREUHEUBUBR4,BR3LGDFR2D2GLHL
    BU2BR4U2BUBR4
3120 REM SET UP CHARACTER ARRAYS
3130 DIM LE$(26)
3140 FOR K9=Ø TO 26:READ LE$(K9):NEXT
3150 FOR K9=Ø TO 9:READ NU$(K9):NEXT
3170 RETURN
3180 REM ROUTINE TO PRINT A$
3190 FOR K9=1 TO LEN(A$)
3200 B$=MID$(A$,K9,1)
3210 IF B$>="Ø" AND B$<="9" THEN
    DRAW NU$(VAL(B$)):GOTO 324Ø
3220 IF B$="□" THEN N9=Ø ELSE
    N9=ASC(B$)-64
3230 DRAW LE$(N9)
3240 NEXT
3250 RETURN
```

The text window displays the current orders to that unit, and it changes colour on screen. The player is then asked DO YOU WANT TO CHANGE ORDERS (Y/N)?

The Dragon/Tandy program has additional lines (from 3ØØØ to 325Ø) for drawing letters and numbers on the high resolution graphics screen.

ISSUING ORDERS

This routine displays the order options if the player wishes to change the orders given to a particular unit.

```
1900 REM Select Action
1910 GOSUB 254Ø
1920 PRINT AT 18,Ø;"Options are:"
1930 FOR j=1 TO 4
1940 PRINT AT 17+j,8; o$(j,1);
    " - ";o$(j)
1950 NEXT j
1960 REM Dummy for loop
1962 LET a=Ø
1965 LET f$="FfHhMmSs"
1970 LET g$=INKEY$: IF g$="" THEN
    GOTO 197Ø
1975 FOR k=1 TO 8
1980 IF f$(k)=g$ THEN LET a=INT
    ((k+1)/2)
1985 NEXT k
1990 IF a <=Ø THEN GOTO 196Ø
2000 IF i<>6 AND i<>5 AND a=1
    THEN GOSUB 254Ø: PRINT AT 18,8;"No
    bows" : GOSUB 241Ø: GOTO 191Ø
2010 IF a=4 THEN GOSUB 244Ø: RETURN
2020 LET T(i,1)=a
2030 IF a=3 THEN GOSUB 2Ø5Ø
2040 RETURN
```

```
1900 REM SELECT ACTION
1910 GOSUB 254Ø
1920 PRINT "OPTIONS ARE:□"
1930 FOR J=Ø TO 3
1940 PRINT LEFT$(O$(J),1);"□-□"; O$(J)
1950 NEXT J
1960 REM DUMMY FOR LOOP
1965 F$="FHMS"
1970 GET G$:IF G$="" THEN 197Ø
1973 A=-2
1975 FOR K=1 TO 4
1980 IF MID$(F$,K,1)=G$ THEN A=K-1
1985 NEXT K
1990 IF A<=-1 THEN 196Ø
2000 IF I<>4 AND I<>5 AND A=Ø
    THENGOSUB254Ø:PRINT "NO BOWS":
    GOSUB241Ø:GOTO 191Ø
2010 IF A=3 THEN GOSUB 244Ø:RETURN
2020 T(I,Ø)=A
2030 IF A=2 THEN GOSUB 2Ø5Ø
2040 RETURN
```

```
1900 DEF PROCactsel
1910 PROCclean
1920 PRINT TAB(2,23);"Options are:"
1930 FOR j=ØTO3
1940 PRINT TAB(12,24+j); LEFT$(order$(j),
    1);"□-□";order$(j)
1950 NEXT
1960 REPEAT
1970 g$=GET$
1980 A%=(INSTR("FfHhMmSs",
    g$)+1) DIV 2-1
1990 UNTIL A%>-1
2000 IF i<>4 AND i<>5 AND A%=Ø
    THEN PRINT"No bows":GOTO 192Ø
2010 IFA%=3 THEN PROCstat: ENDPROC
2020 T%(i,Ø)=A%
2030 IF A%=2 THEN PROCww
2040 ENDPROC
```

```
1900 REM SELECT ACTION
1910 GOSUB 254Ø
1920 DRAW"BMØ,152":A$="OPTIONS□□
    □□":GOSUB 319Ø
1930 FOR J=1 TO 4
1940 DRAW"BM1Ø4,"+STR$(144+J*8):
    A$=LEFT$(O$(J),1)+"□□"+O$(J):
    GOSUB 319Ø
1950 NEXT J
1960 REM
1962 A=Ø
1965 F$="FHMS"
1970 G$=INKEY$:IF G$="" THEN 197Ø
1980 A=INSTR(1,F$,G$)
1990 IF A<=Ø THEN 196Ø
2000 IF I<>6 AND I<> 5 AND A=1
    THEN GOSUB 254Ø:DRAW"BM64,152":
    A$="NO BOWS":GOSUB 319Ø:
    GOSUB 241Ø:GOTO 191Ø
2010 IF A=4 THEN GOSUB 244Ø: RETURN
2020 T(I,1)=A
2030 IF A=3 THEN GOSUB 2Ø5Ø
2040 RETURN
```

Lines 1920 to 1950 display the four options. Lines 1965 to 1985 read the keyboard, and set A (or a) equal to the order number—0 is Fire, 1 is Halt, 2 is Move and 3 is Status.

The Fire option will be dealt with in the next part of Cavendish Field. The Move and Status options will be dealt with below. There is no routine for Halt, because the Move routine is simply by-passed. The direction element in the troop array is not reset because, as you will see later, it is used in the hand-to-hand combat routine.

Line 2020 feeds the value of A into the troop array. The order number is placed in the first element of the array.

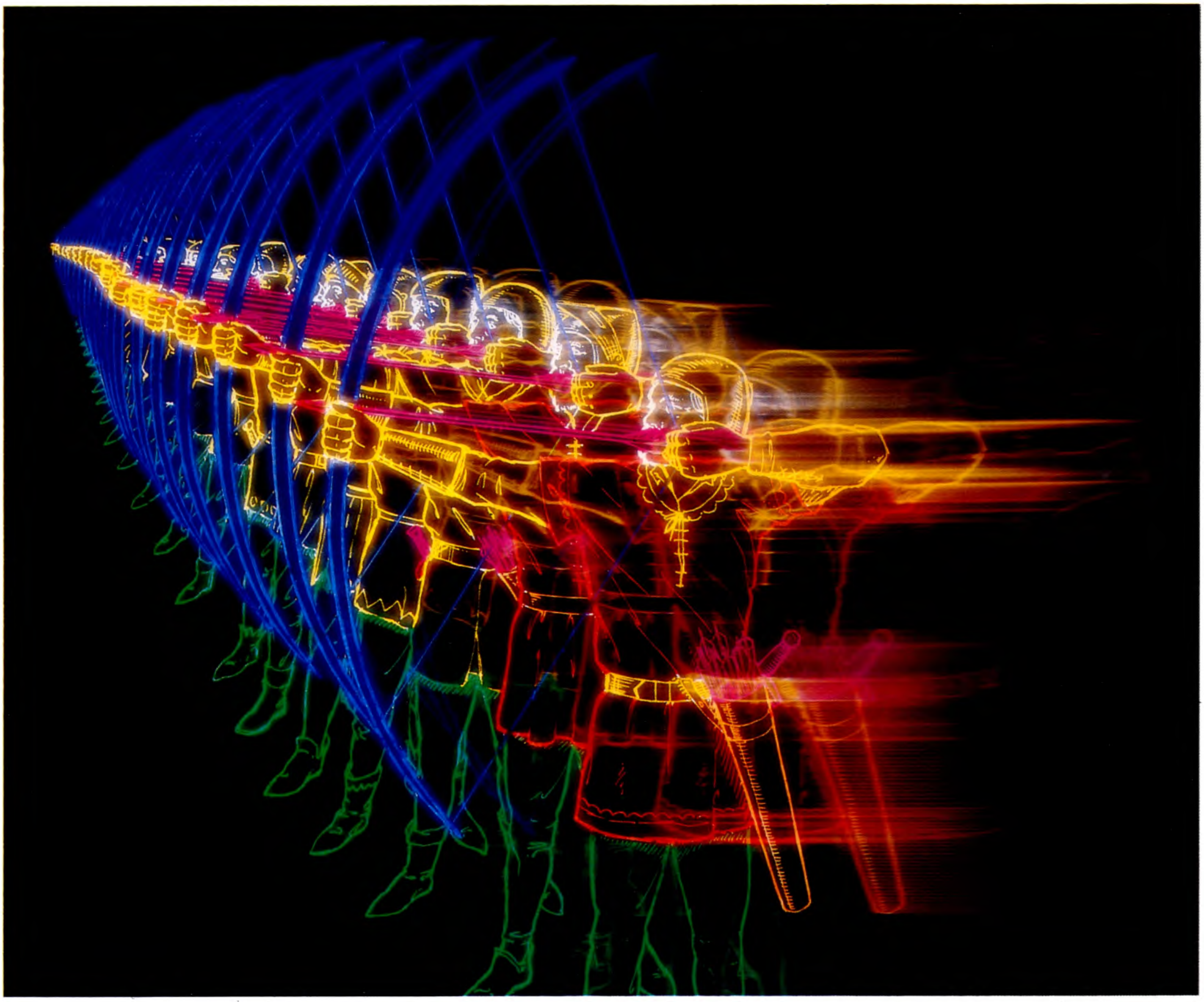
A NEW DIRECTION

If the player issues an order to move, a direction must be given also. This routine handles the movement options.

```
2050 REM Decide Movement Direction
2055 GOSUB 2540
2060 PRINT AT 17,0;"Which way (NSEW)?"
2065 LET g=0
2070 REM Dummy for loop
2080 LET g$=INKEY$: IF g$="" THEN
   GOTO 2080
2090 IF CODE (g$)>90 THEN LET
   g$=CHR$ (CODE (g$)-32)
2095 FOR k=1 TO 4
2100 IF i$(k)=g$ THEN LET g=k
2105 NEXT k
2110 IF g=0 THEN GOTO 2070
2120 LET T(i,2)=g
2130 RETURN
```

```
2050 REM DECIDE MOVEMENT DIRECTION
2055 GOSUB2540
2060 PRINT "WHICH WAY? (NSEW)"
2065 G=0
2070 REM DUMMY FOR REPEAT LOOP
2080 GET G$:IFG$=""THEN 2080
2090 IF ASC(G$)>90 THEN
   G$=CHR$(ASC(G$)-32)
2095 FOR K=1 TO 4
2100 IF MID$(I$,K,1)=G$ THEN G=K
2105 NEXT K
2110 IF G=0 THEN 2070
2120 T(I,1)=G
2130 RETURN
```

```
2050 DEF PROCww
2060 PRINT"Which way (NSEW)?"
2070 REPEAT
2080 g$=GET$
2090 IF ASC(g$)>90 THEN
```



```
    g$=CHR$(ASC(g$)-32)
2100 g=INSTR(dir$,g$)
2110 UNTIL g
2120 T%(i,1)=g
2130 ENDPROC
```

```
2050 REM DECIDE MOVEMENT DIRECTION
2055 GOSUB 2540
2060 DRAW"BM0,144":A$="WHICH WAY
    □□N□S□E□W":GOSUB 3190
2065 G=0
2070 REM
2080 G$=INKEY$:IF G$="" THEN 2080
2100 G=INSTR(1,I$,G$)
2110 IF G<=0 THEN 2070
2120 T(I,2)=G
2130 RETURN
```

The routine checks that the letter input by the player is either N, S, E or W. If it is, then Line 2120 feeds the move direction code into the second element of the troop array.

STATUS

If you want to find out the status of any key unit while planning the big push, you can select the Status option.

```
2440 REM Unit status
2450 GOSUB 2540
2460 PRINT AT 17,0;"UNIT□";i;"□□□□
    Type:□";t$(i)
2470 PRINT AT 18,0;"Weapon:□";w$(T(i,3))
2480 PRINT AT 18,15;"Armour:□";a$(T(i,4))
2490 PRINT AT 19,0;"Strength:□";T(i,7)
2500 PRINT AT 19,14;"Attitude:";m$(T(i,5))
2510 PRINT AT 20,0;"Location:□";r$
    (m(T(i,8),T(i,9))+1)
2520 GOSUB 2410
2530 RETURN
```

```
2440 REM STATUS OF UNIT
2450 GOSUB 2540
2460 PRINT"UNIT□";I+1;
    "□TYPE:□";T$(I)
2470 PRINT "WEAPON:□";W$(T(I,2));
    "□□□□□□ARMOUR:□";A$
    (T(I,3))
2480 PRINT "STRENGTH:□";T(I,6)
2500 PRINT "LOCATION:□";R$(M(T(I,7),
    T(I,8)));"□□□ATTITUDE:□";
    M$(T(I,4))
2520 GOSUB 2410
2530 RETURN
```

```
2440 DEF PROCstat
2450 PROCclean
2460 PRINTTAB(2,23); "Unit□"; i+1;
    "□□□□ Type: □"; type$(i)
2470 PRINTTAB(2,24); "Weapon: □"; wpn$
    (T%(i,2))
2480 PRINTTAB(2,25); "Armour: □"; arm$
    (T%(i,3))
2490 PRINTTAB(2,26); "Current strength: □";
    T%(i,6)
2500 PRINTTAB(2,27); "Located in □"; ter$
    (FNmread (T% (i,7), T%(i,8)))
2510 PRINTTAB(2,28); "Attitude is □"; mor$
    (T%(i,4))
2520 PROCpause
2530 ENDPROC
```

```
2440 REM UNIT STATUS
2450 GOSUB 2540
2460 DRAW"BM0,144":A$="UNIT"+STR$
    (I)+"□□□□TYPE□"+T$(I):
    GOSUB 3190
2470 DRAW"BM0,152":A$="WEAPON
    □"+W$(T(I,3)):GOSUB 3190
2480 DRAW"BM120,152":A$="ARMOUR
    □"+A$(T(I,4)):GOSUB 3190
2490 DRAW"BM0,160":A$=
    "STRENGTH"+STR$(T(I,7)):GOSUB 3190
2500 DRAW"BM120,160":A$="ATTITUDE
    □"+M$(T(I,5)):GOSUB 3190
2510 DRAW"BM0,168":A$="LOCATION
    □"+R$(M(T(I,8),T(I,9))+1):
    GOSUB 3190
2520 GOSUB 2410
2530 RETURN
```

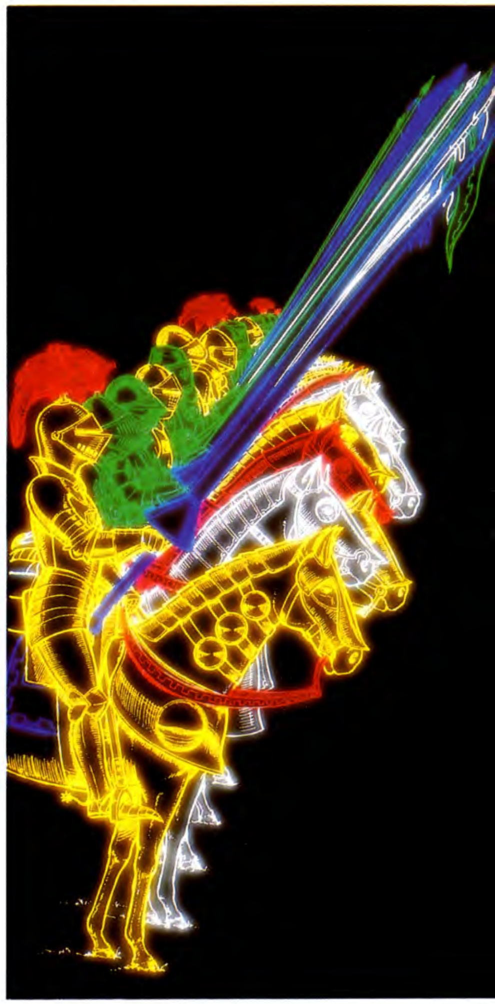
All the elements in the troop array (or their equivalents in words) are displayed.

THE EFFECT OF ORDERS

The routine tells the player if a particular unit is complying with the order.

```
1020 REM Effect of orders
1030 FOR i=1 TO 16
1032 IF t(i,1)>3 THEN GOTO 1140
1035 INK 0: GOSUB 2540: PRINT AT
    17,0;"Unit□";i;"□decides to act"
1040 LET cl=1: IF i>8 THEN LET cl=2: INK
    cl
1050 IF T(i,1)=3 THEN LET b=i: GOSUB
    1160
1055 IF T(i,1)=2 THEN GOTO 1140
1060 IF T(i,1)=1 THEN LET sh=i: GOSUB
    1710: GOTO 1140
1070 FOR f=-1 TO 1
1080 FOR g=-1 TO 1
1090 FOR e=1 TO 16
1100 IF (T(i,8)+f=T(e,8)) AND
    (T(i,9)+g=T(e,9)) AND T(e,1)<>5
    THEN LET us=i: LET th=e: GOSUB 1510
1110 NEXT e
1120 NEXT g
1130 NEXT f
1140 NEXT i
1150 RETURN
```

```
1020 REM EFFECT OF ORDERS
1030 FOR I=0 TO 15
1032 IF T(I,0)>2 THEN 1140
1034 GOSUB2540:PRINT"UNIT□";I+1;"□
    DECIDES TO ACT"
1040 CO=1:IF I>7 THEN CO=9
1050 IF T(I,0)=2 THEN B=I:GOSUB 1160
1055 IF T(I,0)=1 THEN 1140
1060 IF T(I,0)=0 THEN SH=I:GOSUB 1710:
    GOTO 1140
1070 FOR F=-1 TO 1
1080 FOR G= -1 TO 1
1090 FOR E=0 TO 15
1100 IF(T(I,7)+F=T(E,7))AND(T(I,8)+
    G=T(E,8))ANDT(E,0)<>4THEN US=I:
    TH=E:GOSUB1510
1110 NEXT E
1120 NEXT G
1130 NEXT F
1140 NEXT I
1150 RETURN
```

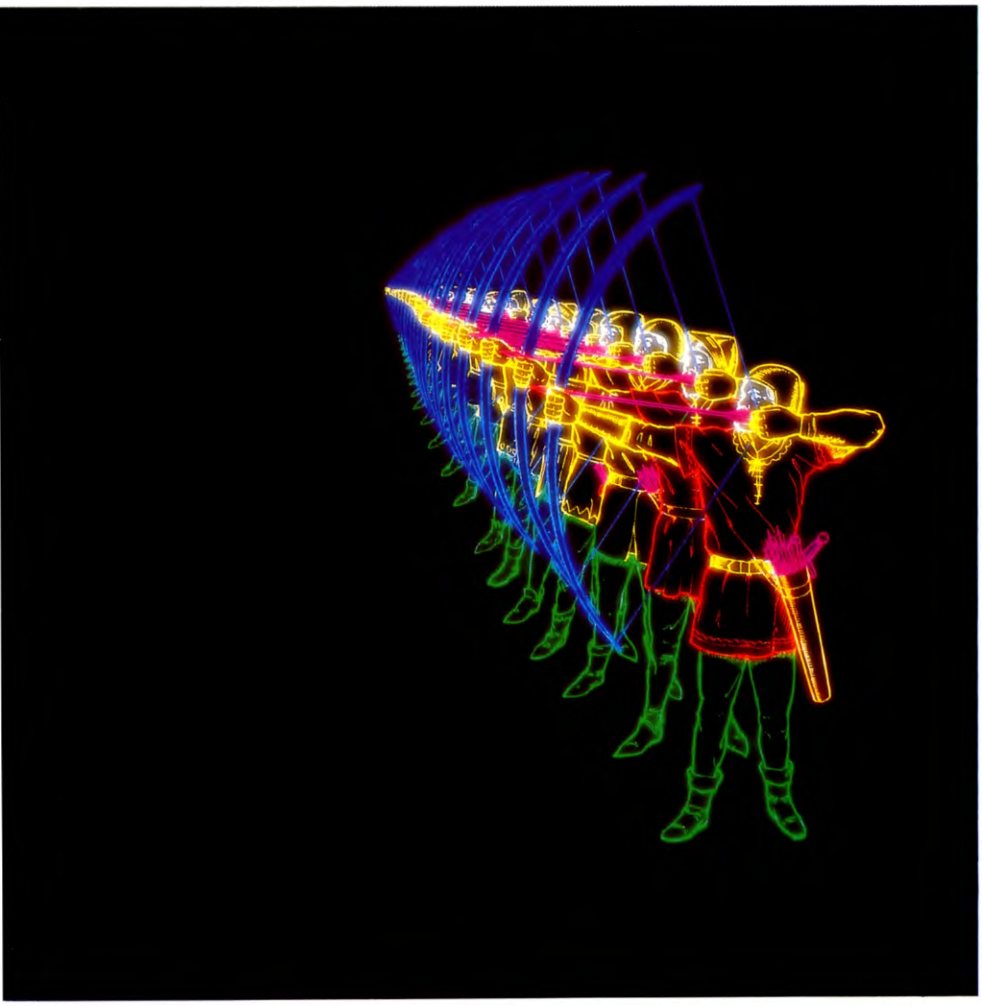


```
1020 DEF PROCeffect
1030 FOR i=Ø TO 15
1031 COLOUR Ø
1032 IF T%(i,Ø)>2 THEN 114Ø
1035 PROCclean:PRINT TAB(2,23);"Unit□";
   i+1;"□decides to act"
1040 cl=(i□DIV 8)+1:COLOURcl
1050 IF T%(i,Ø)=2 THEN PROCmove(i)
1055 IF T%(i,Ø)=1 THEN GOTO114Ø
1060 IF T%(i,Ø)=Ø THEN PROCfire(i):
   GOTO 114Ø
1070 FORfig=-1TO1
1080 FORgif=-1TO1
1090 FORen=ØTO15
1100 IFT%(i,7)+fig=T%(en,7)AND T%
   (i,8)+gif=T%(en,8) AND T%(en,
   Ø)<>4 THENPROCcombat(i,en)
1110 NEXT:NEXT:NEXT
1140 NEXT
1150 ENDPROC
```

```
1020 REM EFFECT OF ORDERS
1030 FOR I=1 TO 16
1032 IF T(I,1)>3 THEN 114Ø
1035 COLOR 4:GOSUB 254Ø:DRAW
   "BMØ,144":A$="UNIT"+STR$(I)+
   "□ACTS":GOSUB 319Ø
1040 CL=3: IF I>8 THEN CL=4:COLOR CL
1050 IF T(I,1)=3 THEN B=I:GOSUB 116Ø
1055 IF T(I,1)=2 THEN 114Ø
1060 IF T(I,1)=1 THEN SH=I:
   GOSUB 171Ø:GOTO 114Ø
1070 FOR F=-1 TO 1
1080 FOR G=-1 TO 1
1090 FOR E=1 TO 16
1100 IF (T(I,8)+F=T(E,8)) AND (T(I,9)+
   G=T(E,9)) AND T(E,1)<>5 THEN
   US=I:TH=E:GOSUB 151Ø
1110 NEXT E,G,F
1140 NEXT I
1150 RETURN
```

The routine operates quite simply. It cycles through each of the sixteen units and acts on its order. It will only allow combat if a unit has move orders, because that unit is then the attacker. If it finds halt orders, it moves on to the next unit, taking no action. If it finds fire orders, it calls the missile routine. If it finds move orders it calls the movement routine.

When movement has been carried out it looks at each map square adjacent to the unit's new position—Lines 1Ø7Ø to 115Ø. If the adjacent square is occupied by an enemy unit the combat routine is called—you will see how to enter this in the next part.



THE COMPUTER OPPONENT

The computer gives orders to its units in a more or less random way. In the last part of this series of articles you'll see how to give your opponent some degree of intelligence, but for the moment, this simplified routine will give you a playable wargame.

```
2140 REM Enemy selects action
2150 LET T(e,2)=3
2160 LET T(e,1)=FN r(3)
2170 IF T(e,1)=1 AND T(e,3)<>2 THEN
   GOTO 216Ø
2180 IF T(e,1)=3 THEN IF FN r(2)=1 THEN
   LET T(e,2)=FN r(4)
2190 RETURN
```

```
2140 REM SELECT ENEMY ACTION
2150 T(E,1)=3
2160 T(E,Ø)=FNR(3)-1
2170 IF T(E,Ø)=Ø AND T(E,2)<>1 THEN
   216Ø
2180 IF T(E,Ø)=2 THEN IF FNR(2)=1 THEN
   T(E,1)=FNR(4-1)
2190 RETURN
```

```
2140 DEF PROCensl
2150 T%(en,1)=3
2160 T%(en,Ø)=RND(3)-1
2170 IF T%(en,Ø)=Ø AND T%(en,2)<>1
   THEN 216Ø
2180 IF T%(en,Ø)=2 THEN IF RND(2)=1
   THEN T%(en,1)=RND(4)-1
2190 ENDPROC
```

```
2140 REM ENEMY SELECTS ACTION
2150 T(E,2)=3
2160 T(E,1)=RND(3)
2170 IF T(E,1)=1 AND T(E,3)<>2 THEN
   216Ø
2180 IF T(E,1)=3 AND RND(2)=1 THEN
   T(E,2)=RND(4)
2190 RETURN
```

The routine generates a random number in Line 216Ø which decides on the action to be taken by each of the computer's units in turn.

If movement is selected, the movement isn't completely random because the units will tend to move south—see Line 215Ø.

A COMPUTER INTERIOR DESIGNER-2

Once you've drawn your room outline, try your hand at designing a few items of furniture then set about perfecting your room plan before saving the lot to tape or disk

This, the second and final part of the room planner program, contains the remaining section of the listing and instructions on how to use the options available. The first three options are described in detail in each machine's section, the other four are standard.

The various options are:

- Option one, design the room plan
- Option two, position furniture
- Option three, design your own furniture
- Option four, save the design
- Option five, load your design
- Option six, print out your design
- Option seven, exit program

The first question you are asked in option one is the maximum dimension of the room (in metres). Enter this information and press ENTER. To draw the room, press SPACE if you want to draw the walls—otherwise choose (W)indow, (D)oor or (B)lank for any other opening.

Once you've pressed the relevant key, you are then asked for a direction and distance. Key in this information and the questions will be repeated. This is to allow you to enter a diagonal line (by specifying so much up, so much to the right, for instance). If the line is a simple horizontal or vertical, you will already have supplied enough information, so enter Ø to both these questions the second time they are asked; for a diagonal enter a new direction and distance. Continue until the room plan is complete, then press Q to return to the main menu. A few minutes' practice will show you exactly how this works.

In option two, use 6 and 7 to move the flashing arrow opposite the object you wish to select. Press S and the object will appear in the middle of the screen. You can then move it left, down, up or right using keys 5, 6, 7 or 8. To rotate the object, use A for anticlockwise and C for clockwise.

To draw the items in option three, you use the same procedure as the room plan, except the object will not appear until all the dimensions have been entered. The first question you will be asked is the number of sides (1–15) and then you are instructed to enter a two letter identity code. You can then proceed to draw the item.

Option four saves both the screen and the data for the furniture. You can load in a saved design using option five. The print option—option six—will only work if you have a Sinclair printer connected.

To use this program with the Commodore 64, you need to have Simons' basic or the *INPUT* Hi-Res Utility. If you have the latter, you'll need to make a couple of alterations to the listing. In Line 4, the **224** (printed in bold type) changes to 32 and in Line 5, **225** changes to 63.

When you run the program, the main menu will appear on the screen offering seven options:

- Option one, design the room plan
- Option two, position furniture
- Option three, design your own furniture
- Option four, save the design
- Option five, load your design
- Option six, print out your design
- Option seven, exit program

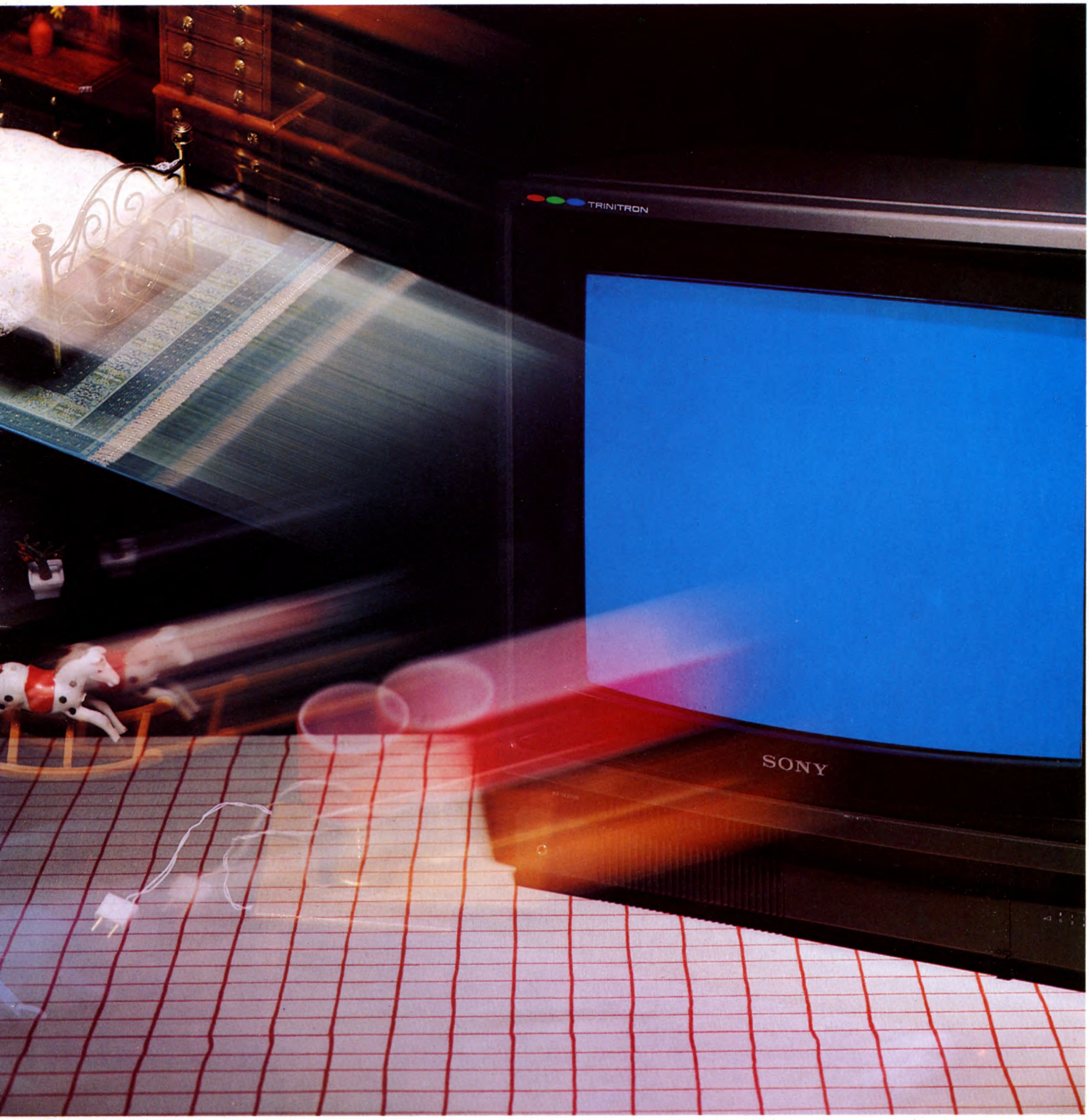
If you press 1 to select the first option, you will be asked for the length of the room in metres; insert the largest dimension, then press RETURN. You are then offered the options of pressing different keys to select what to draw. If you want to depict a wall, press the space bar, otherwise choose (W)indow, (D)oor, or (B)lank—which is for any other type of opening.

Next, you will be asked for a direction and distance. Insert this information, press RETURN and the question will be repeated. If you simply want a vertical or horizontal line press the space bar. A diagonal line needs to be specified in two directions (so much up, so much to the right, for instance) so in this case enter a second direction and distance.

In option two, use the Z, X, ; and / keys to move the flashing asterisk to the item of furniture you wish to select from the menu and press S (for select). Use the same keys to move the asterisk to the proposed site and press P (for place). To adjust its position, press A and Z, X, ; or / to move the item and L to rotate it clockwise or K to rotate it



- COMPLETE THE ROOM OUTLINE
- DESIGN YOUR OWN FURNITURE AND ROOM LAYOUT
- **SAVE** YOUR DESIGNS
- **LOAD** YOUR DESIGNS
- **PRINT** YOUR DESIGNS
- ROTATE YOUR FURNITURE
- DELETE AND REDESIGN
- SELECT A TIME
- EXIT THE PROGRAM




TRINITRON

anticlockwise. To end rotation, press A; you can then straighten any objects with K or L. When you are satisfied with the position of the piece, press Q. To see the object you are carrying press *. Press * again to exit.

To design the furniture in option three, you use the same technique used for drawing the room. No item can be larger than two metres or have more than 8 sides. You will be asked to give your item a number and a two letter code to identify it.

The remaining options allow you to save a design, to save or load a stored program. If you have a suitable screen dump program, you can also print out your design. The final option allows you to quit the program.

When you run the program, the main menu will appear showing the seven options:

- Option one, design the room plan
- Option two, position furniture
- Option three, design your own furniture
- Option four, save the design
- Option five, load your design
- Option six, print out your design
- Option seven, exit program

For the first option, press 1, and you will be asked to enter the maximum length of the room in metres. Key in this information and press RETURN. A dot which indicates the starting point for the room plan will then appear in the lower left hand corner of the graphics window.

When you draw your room you can press different keys and choose to insert a window or door (W) or simply leave a blank space (B). If you want to draw a wall all you need to do is enter a direction (U)p, (D)own, (R)ight or (L)eft and a distance, so U4 (RETURN) will give you a four metre wall drawn upwards on the screen. To draw a diagonal you must enter two directions and distances such as U4L4 RETURN.

If you make a mistake and want to clear the room plan, press C. When you're satisfied with your room plan press F. This will return you to the main menu and you can then go on to option two.

In option two, the room plan you have just created will remain in the graphics window, ready for you to position the furniture. While you are offered a choice of ten items of furniture, only five are predefined. (All of these relate to the kitchen.) The remaining five can be defined as you wish using option three. The predefined items are located at the bottom of the screen and you will need to move them up into the room plan.

To do this, use the cursor keys to position the flashing asterisk in the centre of the chosen item and press S (for select). Now use the same keys to move the asterisk to the point in the room where you want to place the item and press P (for place).

Once the items are in position, you can carry out final adjustments. Move the flashing asterisk to the centre of an item, then press A. You can now use the cursor keys to move the furniture one way or the other. If you want to rotate it, press L or K—or to delete it press COPY with the E key depressed. To fix the item in its final position press COPY. Press Q to return to the main menu.

Although this is listed as the third option, new items of furniture must be designed before you design the room plan, since the furniture planner uses the same graphics space. If you should forget this, you can save your room plan then load it back after designing the furniture.

Each piece must first be given a file number, and also identified by letter code so that you can tell whether it is a cupboard or a table, for instance. To define a new item, enter two letters to define the item, for example, type ta for table. To design the furniture you simply use the same method as designing the room, except all measurements must be entered in centimetres. No item can be larger than two metres (200 cm) square, or have more than eight sides.

The remaining options allow you to save a design to a recorder, or load one back again. If you have a suitable screen dump program you can use this with option six to take a printout of the design. And, finally, there is a quit option.

When you run the program the main menu will appear on the screen showing the seven options:

- Option one, design the room plan
- Option two, position furniture
- Option three, design your own furniture
- Option four, save the design
- Option five, load your design
- Option six, print out your design
- Option seven, exit program

If you choose option one, you will be asked to enter the maximum length of the room in metres. This allows the computer to scale the plan to fit the screen. To start drawing, press the space bar for a black line, B for a blank area, W for a window or O for a door. You will then be asked for a direction and distance.

Once you have entered this information the questions will be repeated. This is to allow you to plot a diagonal line, by specifying so much to the right, so much up, for instance. You only need to enter a new direction and distance if you want a diagonal, otherwise press ENTER. If you make a mistake, press C to clear the plan and F to return to the menu.

With option two, a room plan drawn using option one will remain on the screen, together with a cross-shaped cursor. This can be moved around pixel-by-pixel using the arrow keys. Pressing SHIFT with the arrow keys will make the cursor move quicker. When the cursor is in the correct place, press P, and you will be asked for the number of the item you wanted sited there—Ø = sink, 1 = cooker, 2 = table, 3 = fridge and 4 = cupboard. You are given a choice of numbers Ø–9, but only Ø–4 are pre-defined in this option, and the remaining furniture can be defined as you want using option three. To delete an object, put the cross back on the corner and press D followed by the number of the object. Make sure you position the cross accurately, otherwise the object will not be totally deleted.

You can rotate an item, by pressing R. You will then be asked to enter a rotation angle between Ø and 36Ø. Once you've done this, all the objects drawn will be at this angle until you press R once again—Ø is the default setting. To return to the main menu press F.

Option three allows you to design your own items, though they must not be more than two metres deep or wide.

First of all you're asked the number of the items, then you can design it in the same way as designing the room. If you make a mistake, press C to start again. Once you've finished, press F to return to the main menu. All distances in this option must be given in centimetres.

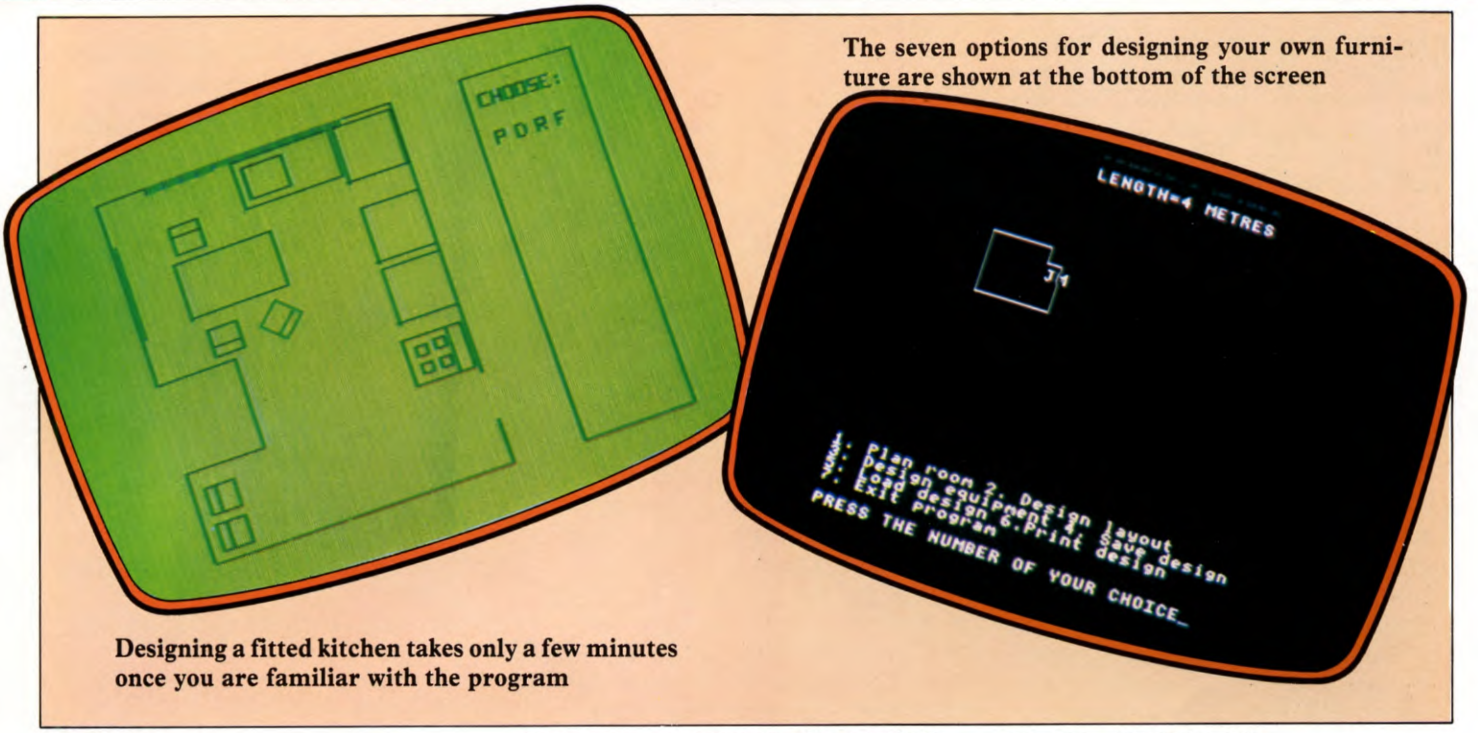
The remaining options allow you to save a design or load in one you have previously recorded. If you have a suitable screen dump program you can choose to make a printout, too. There is also a quit option.

```
7Ø1Ø INPUT "ENTER DIRECTION
   (U,D,L,R)? ";D$
7Ø2Ø INPUT "ENTER DISTANCE? ";D:
   RETURN
7Ø4Ø LET Z=CODE INKEY$: FOR K=1 TO
   LEN K$: IF CODE (K$(K))=Z THEN LET
   K=LEN K$: GOTO 7Ø6Ø
7Ø5Ø NEXT K: GOTO 7Ø4Ø
7Ø6Ø NEXT K: RETURN
8ØØØ DIM S(1Ø): LET NF=Ø: LET
   F$="ØØØ": LET MAX=5: LET R=Ø: LET
   SC=175/MAX: GOSUB 6Ø6Ø
8Ø1Ø DIM O(1Ø,3Ø): DIM O$(1Ø,2)
8Ø2Ø FOR N=1 TO 5: FOR K=1 TO 8: READ
   O(N,K): NEXT K: NEXT N
8Ø3Ø FOR N=1 TO 1Ø: READ O$(N): NEXT N
8Ø32 FOR N=USR "A" TO USR "A"+7:
```

```
READ A: POKE N,A: NEXT N
8Ø35 FOR N=1 TO 1Ø: READ S(N):NEXT N
8Ø4Ø RETURN
8Ø9Ø DATA .5,Ø,Ø,-.5,-.5,Ø,Ø,.5
81ØØ DATA .75,Ø,Ø,-.5,-.75,Ø,Ø,.5
811Ø DATA .5,Ø,Ø,-.75,-.5,Ø,Ø,.75
812Ø DATA .5,Ø,Ø,-.5,-.5,Ø,Ø,.5
813Ø DATA .5,Ø,Ø,-.5,-.5,Ø,Ø,.5
815Ø DATA "CU","CO","WA","SI","FR",
   "XX","XX","XX","XX","XX"
816Ø DATA 16,32,64,255,64,32,16,Ø
817Ø DATA 4,4,4,4,4,Ø,Ø,Ø,Ø,Ø
```

```
173Ø GOSUB 3ØØØØ
174Ø OX=16Ø:OY=14Ø:X=OX:Y=OY
175Ø HIRES 3,6:CSET(Ø)
177Ø PRINT "♥ENTER THE NO OF THE ITEM
   YOU WISH TO":INPUT "DEFINE";Z
1775 IF Z<6 OR Z>1Ø THEN 177Ø
1776 Z=2*Z-1
178Ø PRINT "■ENTER A TWO LETTER CODE
   FOR THE ITEM":INPUT H$((Z+1)/2)
179Ø PRINT "■ENTER MAXIMUM LENGTH
   OF ROOM":INPUT "(METRES)";LE
1795 IF LE<1 THEN PRINT "♥":GOTO
   179Ø
18ØØ CSET(2):TEXT Ø,Ø,"LENGTH□=□2□
   METRES□□□□□□WIDTH□=□2□
   METRES",1,1,8
18Ø5 TEXT 35,1Ø,"ENTER DISTANCES IN
   CENTIMETRES",1,1,8
18Ø6 TEXT 1ØØ,2Ø"8 SIDES MAXIMUN",1,1,8
181Ø SC=189/LE
182Ø NS=.5
183Ø G=1
184Ø IF Y>ØANDY<199ANDX>ØAND
   X<319THENPLOTX,Y,1
1845 DE(Z,G)=Ø:DE(Z+1,G)=Ø:G=G+1
185Ø BLOCK Ø,168,319,199,Ø
188Ø PT=1
194Ø GOSUB 78Ø
195Ø GX=X+NS*(NX(1)+NX(2)):
   GY=Y+NS*(NY(1)+NY(2))
1951 IFGX<ØORGX>319ORGY<ØOR
   GY>199ORX<ØORX>319ORY<ØOR
   Y>199THEN1953
1952 LINE X,Y,GX,GY,PT
1953 X=X+NS*(NX(1)+NX(2)):Y=Y+NS*
   (NY(1)+NY(2))
196Ø DE(Z,G)=(NX(1)+NX(2))/1ØØ:DE
   (Z+1,G)=(NY(1)+NY(2))/1ØØ:G=G+1
1965 IFG>9THEN2Ø2Ø
198Ø BLOCK Ø,168,319,199,Ø
199Ø TEXT Ø,168,"PRESS C TO CLEAR
   EQUIPMENT",1,1,8
1992 TEXT Ø,176,"OR F WHEN
   FINISHED",1,1,8
1993 TEXT Ø,184,"OR <SPACE> TO
   CONTINUE",1,1,8
1994 GET C$:IF C$="" OR(C$<>"□"
   ANDC$<>"F"ANDC$<>"C")
   THEN 1994
2ØØØ IF C$="C" THEN HIRES 3,6:F1=Ø:
   NX(1)=Ø:NX(2)=Ø:NY(1)=Ø:NY(2)=Ø
2ØØ1 IF C$="C" THEN FOR F=1 TO 9:DE
   (Z,T)=Ø:DE(Z+1,T)=Ø:NEXT F:G=1:
   GOTO 18ØØ
2Ø1Ø IF C$="□" THEN 185Ø
2Ø2Ø GOSUB 3ØØ1Ø:RETURN
21ØØ GOSUB 3ØØØØ:CSET(Ø)
21Ø5 NM$="":INPUT "♥ENTER NAME TO
   SAVE■";NM$:IF NM$="" THEN
   CSET(2):RETURN
211Ø GOSUB 2ØØØØ:POKE 24432,LEN(NM$):
   POKE 24388,DV
212Ø FOR Z=1 TO LEN(NM$):POKE
   24432+Z,ASC(MID$(NM$,Z,1)):NEXT Z
213Ø SYS 24379:CSET(2):RETURN
219Ø CSET(Ø)-NM$=""
2195 INPUT "♥ENTER NAME TO
   LOAD■";NM$:IFNM$=""THEN
   CSET(2):RETURN
22ØØ GOSUB 2ØØØØ:LOAD NM$,DV,1:END
229Ø COPY:RETURN
1ØØØØ POKE 198,Ø:WAIT 198,1:PRINT
   "♥":CSET(Ø):C$=""
1ØØØ5 INPUT C$:CSET(2):RETURN
2ØØØØ INPUT "♥(T)APE OR (D)ISK■";IN$
2ØØ1Ø IF IN$="D" THEN DV=8:RETURN
2ØØ2Ø IF IN$="T" THEN DV=1:RETURN
2ØØ3Ø GOTO 2ØØØØ
3ØØØØ POKE 24346,251:POKE 24348,253:SYS
   2432Ø:RETURN
3ØØ1Ø POKE 24346,253:POKE 24348,251:SYS
   2432Ø:RETURN
3ØØ2Ø POKE 198,Ø:WAIT 198,1:POKE
   198,Ø:RETURN
```

```
131Ø DEF PROCselect
132Ø i=Ø
133Ø FOR J=1 TO 3Ø
134Ø IF ABS(take(1)-put(J,1))<=Ø.25*
   scale AND ABS(take(2)-put(J,2))<=
   Ø.25*scale THEN i=put(J,3):J=3Ø:GOTO
   136Ø
1345 NEXT
135Ø FOR J=1 TO 1Ø:IF ABS(take
   (1)-32-J*11Ø)<=4Ø AND ABS
   (take(2)-1ØØ)<=3Ø THEN i=J:J=1Ø
136Ø NEXT:IF i<>Ø THEN SOUND1,-15,
   128,5
137Ø ENDPROC
138Ø DEF PROCputdown
139Ø VDU 7
14ØØ PROCplace(take(1),take(2),i)
141Ø J=Ø
142Ø REPEAT
143Ø J=J+1
144Ø UNTIL put(J,3)=Ø OR J=3Ø
145Ø put(J,1)=take(1):put(J,2)=take(2):put
   (J,3)=i:put(J,4)=phi:i=Ø
146Ø ENDPROC
147Ø DEF PROCplace(e,f,g)
148Ø MOVE e-Ø.2*scale,f:VDU 5:
   PRINTh$(g):VDU 4:MOVEe,f:g=g*2-1
149Ø PLOTØ,def(g,1)*c-def(g+1,1)*s,def
   (g,1)*s+def(g+1,1)*c
1495 FOR TG=2 TO 9
15ØØ PLOT9,def(g,TG)*c-def(g+1,TG)*s,def
   (g,TG)*s+def(g+1,TG)*c
151Ø NEXT
152Ø ENDPROC
153Ø DEF PROCadjustment
154Ø FOR H=1 TO 3Ø
155Ø IF ABS(take(1)-put(H,1))<=Ø.25*
   scale AND ABS(take(2)-put(H,2))<=
   Ø.25*scale THEN j=put(H,3):ax=put(H,1):
   ay=put(H,2):phi=put(H,4):h=H:H=31
156Ø NEXT:IF H<>32 THEN ENDPROC
157Ø c=scale*COS(phi):s=scale*SIN(phi)
158Ø REPEAT
159Ø PROCplace(ax,ay,j)
16ØØ IFINKEY(-26) ax=ax-8
161Ø IF INKEY(-122) ax=ax+8
162Ø IF INKEY(-58) ay=ay+8
163Ø IF INKEY(-42) ay=ay-8
164Ø IF INKEY(-71) phi=phi+RAD(5)
165Ø IF INKEY(-87) phi=phi-RAD(5)
166Ø IF INKEY(-35) GCOL3,Ø:flag2=1
   ELSE GCOL3,1
167Ø c=scale*COS(phi):s=scale*SIN(phi)
168Ø PROCplace(ax,ay,j)
169Ø UNTIL INKEY(-1Ø6)
17ØØ IF flag2=Ø THEN put(h,1)=ax:put
   (h,2)=ay:put(h,3)=j:put(h,4)=phi□ELSE
   FOR TG=1 TO 4:put(h,TG)=Ø:NEXT
171Ø phi=Ø:c=scale:s=Ø:flag2=Ø
172Ø ENDPROC
173Ø DEF PROCownequipment
174Ø CLS:CLG
175Ø INPUT"Enter the number of the item you
   want to define",Z:IF Z<1 OR Z>1Ø
   THEN PRINT"Re-";:GOTO 175Ø
176Ø INPUT"Enter a two letter I.D. for the
   item"h$(Z):h$(Z)=LEFT$(h$(Z),2)
177Ø put(Z,3)=Z:Z=Z*2-1
178Ø CLS:INPUT"Enter maximum length of
   room in METRES"length
179Ø VDU5:MOVE3ØØ,1Ø23:PRINT"LENGTH
   AND WIDTH=2 METRES":VDU 4
18ØØ scale=79Ø/length
181Ø nscale=79Ø/2ØØ
182Ø G=1
183Ø PROClines(2)
184Ø CLS:CLG:ENDPROC
21ØØ DEF PROCsave
21Ø5 PROCpoint(take(1),take(2))
211Ø CLS
212Ø VDU 28,Ø,31,39,28
213Ø PRINTTAB(5,31)"FILE NAME WILL BE
   SCREEN"
```



The seven options for designing your own furniture are shown at the bottom of the screen

Designing a fitted kitchen takes only a few minutes once you are familiar with the program

```
2140 VDU 7
2145 !&7881 = scale
2150 *SAVE"SCREEN"5800□7890
2160 CLS
2180 ENDPROC
2190 DEF PROCload
2200 CLS:CLG
2210 VDU 28,0,31,39,28
2220 VDU 7
2230 *LOAD"SCREEN"
2235 scale = !&7881
2240 CLS
2260 ENDPROC
2270 DEF PROCprint
2280 CLS
2285 VDU26
2290 REM ADD SCREEN DUMP
2295 VDU 24;0;199;1279;1023;28,0,31,39,28
2300 PRINT TAB (5,29) "PRESS ANY KEY
   TO RETURN TO MAIN MENU";: M$ =
   GET$
2310 CLS
2330 ENDPROC
```

```
1210 OX = X:OY = Y
1220 IF I$ = CHR$(8) AND X > 0 THEN X =
   X-1
1230 IF I$ = CHR$(9) AND X < 185 THEN
   X = X + 1
1240 IF I$ = CHR$(94) AND Y > 0 THEN Y =
   Y-1
1250 IF I$ = CHR$(10) AND Y < 185 THEN
   Y = Y + 1
1260 IF I$ = CHR$(21) AND X > 7 THEN X =
   X-8
1270 IF I$ = CHR$(93) AND X < 177 THEN
   X = X + 8
1280 IF I$ = CHR$(95) AND Y > 7 THEN Y =
   Y-8
1290 IF I$ = CHR$(91) AND Y < 177 THEN
   Y = Y + 8
1300 PUT(OX-3,OY-3)-(OX+3,OY+3),
   S,PSET
1310 IF I$ = "P" THEN 1360
1320 IF I$ = "D" THEN 1430
1330 IF I$ = "R" THEN 1500
1340 IF I$ = "F" THEN FORK = 1TO4:
   PCOPYK□TOK + 4:NEXT:GOTO90
1350 GOTO 1160
1360 SOUND 190,1
1370 COLOR0:GOSUB 340
1380 I$ = INKEY$:IF I$ < "0" OR I$ > "9"
   THEN 1380
1390 COLOR1:GOSUB 340
1400 N = VAL(I$)
1410 COLOR0,1:A$ = O$(N):
   GOSUB 1540
1420 GOTO 1160
1430 SOUND200,1
1440 COLOR0:GOSUB 340
1450 I$ = INKEY$:IF I$ < "0" OR I$ > "9"
   THEN 1450
1460 COLOR1:GOSUB 340
1470 N = VAL(I$)
1480 COLOR1,1:A$ = O$(N):
   GOSUB 1540
1490 GOTO 1160
1500 CLS:PRINT "ANGLE OF ROTATION
   (0-360)":INPUT RT
1510 IF RT < 0 OR RT > 360
   THEN 1500
1520 RT = (RT/180)*3.141
1530 SCREEN1,0:GOTO1160
1540 IF A$ = "" THEN RETURN
1550 P = 1:X1 = X:Y1 = Y
1560 B$ = MID$(A$,P,1)
1570 D$ = MID$(A$,P + 1,1)
1580 D = VAL(MID$(A$,P + 2,INSTR(P,A$,
   ";") - (P + 2)))
1590 P = INSTR(P,A$,";") + 1
1600 D = D/100:D2 = FNA(D):
   OX = X:OY = Y
1610 IF D$ = "R" THEN XA = D2*COS(RT):
   YA = -D2*SIN(RT)
1620 IF D$ = "L" THEN XA = -D2*COS(RT):
   YA = D2*SIN(RT)
1630 IF D$ = "D" THEN XA = D2*SIN(RT):
   YA = D2*COS(RT)
1640 IF D$ = "U" THEN XA = -D2*
   SIN(RT):YA = -D2*COS(RT)
1650 XA = INT(XA + .5):YA = INT(YA + .5)
1660 X = X + XA:Y = Y + YA
1670 IF B$ = "D" THEN LINE(OX,OY) - (X,
   Y),PSET
1680 IF P < LEN(A$)THEN 1560
1690 X = X1:Y = Y1:RETURN
1700 CLS:LINE INPUT "FILENAME:";F$
1710 SG = PEEK(188)*256
1720 CLS:PRINT"SAVING:";F$
1730 CSAVEM F$,SG,SG + 6143,35252
1740 GOTO 90
1750 CLS:LINE INPUT "FILENAME:";F$
1760 PRINT"PLEASE START TAPE & WAIT"
1770 CLOADM F$
1780 GOTO 90
1790 REM ADD SCREEN DUMP ROUTINE
1795 GOTO 90
```

DESPERATE DECORATOR

This will teach you to use the right paint! As the runs start to form, your carpet is right in the way and you'll have to wield your roller swiftly if you're going to save it

Getting drips of paint on the carpet is one of the nightmares of the do-it-yourself decorator. In this game we give you the opportunity to improve your drip-stopping technique without the damage.

The aim of the game is to prevent the runs of paint dripping down the screen (the wall) and reaching the carpet, using your paint roller. You control the position of the 'roller' graphic on screen, using the keyboard.

In order to perform the calculations quickly enough, the program contains machine code. There are checksums to catch any DATA errors but it is still wise to SAVE the program before you RUN it, in case of any mistakes.

```
1 CLEAR 28671: GOSUB 100
5 CLS : PRINT AT 8,2;" ENTER LEVEL OF
   DIFFICULTY"'TAB8;"<1>□EASY"'TAB
   8;"<2>□FAIRLY EASY"'TAB 8;
   "<3>□NORMAL"'TAB 8;"<4>□
   DIFFICULT"'TAB 8;"<5>□
   IMPOSSIBLE"
6 LET D$=INKEY$: IF D$<"1" OR
   D$>"5" THEN GOTO 6
7 POKE 28951,((D$="1")*200)+((D$=
   "2")*175)+((D$="3")*80)+((D$=
   "4")*40)+(D$="5")
10 BORDER 0: PAPER 7: INK 2: CLS : LET
   a$="": FOR n=1 TO 32: LET
   a$=a$+"□": NEXT n
14 POKE 28953,0: FOR n=1 TO 4: PRINT
   PAPER 2;a$;: NEXT n
15 FOR N=19 TO 21: PRINT PAPER 6;AT
   N,0;A$: NEXT N
16 PLOT 0,143: DRAW 255,0
20 RANDOMIZE USR 28672: RANDOMIZE
   USR 28702
40 PRINT AT 12,7; FLASH 1; PAPER 5; INK
   0;"□G□A□M□E□O□V□E□R□
   "'; FLASH 0;AT 14,7;PAPER 7;"FINAL
   SCORE□";: LET B$="": FOR
   N=28945 TO 28950: LET
   B$=B$+CHR$ (PEEK N): NEXT N:
   PRINT B$
45 FOR N=1 TO 500: NEXT N
50 IF INKEY$="" THEN GOTO 50
60 RUN 5
100 LET L=500: RESTORE L: FOR N=28672
   TO 28961 STEP 8
110 LET T=0: FOR D=0 TO 7: READ A:
   POKE N+D,A: LET T=T+A: NEXT D
120 READ A: IF A<>T THEN PRINT "DATA
   ERROR AT LINE□";L: STOP
130 LET L=L+10: NEXT N: RETURN
500 DATA 33,34,113,6,0,62,32,119,399
510 DATA 35,16,252,33,128,100,34,32,630
520 DATA 113,33,48,48,34,17,113,34,440
530 DATA 19,113,34,21,113,201,205,228,934
540 DATA 112,89,22,0,33,34,113,25,428
550 DATA 126,60,254,156,200,245,229,205,1475
560 DATA 176,34,209,193,245,126,254,255,1492
570 DATA 40,2,120,18,241,254,0,71,746
580 DATA 62,128,40,4,203,31,16,252,736
590 DATA 70,176,119,58,25,113,214,64,839
600 DATA 50,25,113,194,194,112,237,75,1000
610 DATA 32,113,62,0,33,26,113,197,576
620 DATA 205,214,112,193,62,223,219,254,1482
630 DATA 245,203,31,203,31,48,16,203,980
640 DATA 31,48,15,203,31,48,14,203,593
650 DATA 31,48,13,241,195,156,112,12,808
660 DATA 24,7,4,24,4,5,24,1,93
670 DATA 13,241,121,254,240,48,13,120,1050
680 DATA 254,150,48,8,254,32,56,4,806
690 DATA 237,67,32,113,237,75,32,113,906
700 DATA 197,33,29,113,205,214,112,193,1096
710 DATA 121,230,248,79,89,22,0,33,822
720 DATA 34,113,25,72,6,24,126,185,585
730 DATA 32,5,61,119,205,252,112,35,821
740 DATA 16,244,42,23,113,45,32,253,768
750 DATA 37,242,197,112,62,127,219,254,1250
760 DATA 203,31,218,30,112,201,229,120,1144
770 DATA 205,176,34,235,225,1,3,0,879
780 DATA 237,176,235,201,42,118,92,237,1338
790 DATA 91,120,92,25,237,90,84,93,832
800 DATA 41,41,25,41,41,41,25,34,289
810 DATA 118,92,76,201,213,245,17,22,984
820 DATA 113,26,60,254,58,32,6,62,611
830 DATA 48,18,27,24,244,18,241,209,829
840 DATA 201,48,48,48,48,48,48,200,689
850 DATA 0,64,0,0,0,255,255,255,829
860 DATA 128,100,36,36,37,37,35,35,444
```

The roller is under keyboard control on the Y, U, I and O keys. The machine code handles the roller movement, the drips and the score.

In the BASIC program, Line 1 makes space for the machine code and calls lines 100 to 130 to read in the machine code from the DATA (Lines 500–860). Lines 5–7 set the difficulty level and POKE the delay variables accordingly. Lines 10–16 set the screen and Line 20 calls the machine code. Lines 40–60 end the game and determine the score.

```
10 DATA 162,0,169,0,157,0,203,232,208,250,
   #1381
15 DATA 169,5,141,32,208,32,108,193,162,0,
   #1050
20 DATA 169,6,157,192,7,232,224,40,208,
   248,#1483
25 DATA 96,173,186,3,141,184,3,172,185,3,
   #1146
30 DATA 173,182,3,133,251,173,202,3,141,
   203,#1464
35 DATA 3,173,183,3,133,252,32,248,192,169,
   #1388
40 DATA 0,145,251,165,197,201,50,208,47,
   56,#1320
45 DATA 165,251,233,1,206,203,3,133,251,
   165,#1611
50 DATA 252,233,0,133,252,206,184,3,173,
   184,#1620
55 DATA 3,201,0,208,21,169,8,141,184,3,
   #938
60 DATA 165,251,233,56,133,251,165,252,
   233,1,#1740
65 DATA 201,32,144,108,133,252,165,197,
   201,55,#1488
70 DATA 208,47,24,165,251,105,1,238,203,3,
   #1245
75 DATA 133,251,165,252,105,0,133,252,238,
   184,#1713
80 DATA 3,173,184,3,201,9,208,21,169,1,
   #972
85 DATA 141,184,3,165,251,105,55,133,251,
   165,#1453
90 DATA 252,105,1,201,62,176,55,133,252,
   173,#1410
95 DATA 141,2,201,2,208,5,152,56,233,8,
   #1008
100 DATA 168,173,141,2,201,1,208,5,152,24,
   #1075
105 DATA 105,8,168,169,255,145,251,140,
   185,3,#1429
```

- THE PROGRAM
- CATCHING THE DRIPS
- KEYBOARD CONTROL
- CHECKING THE SCORE
- MOPPING UP



```
110 DATA 173,203,3,141,202,3,173,184,3,
   141,#1226
115 DATA 186,3,165,251,141,182,3,165,252,
   141,#1489
120 DATA 183,3,169,0,141,201,3,169,19,141,
   #1029
125 DATA 4,212,165,252,141,1,212,169,18,
   141,#1315
130 DATA 4,212,32,248,192,76,29,193,162,0,
   #1148
135 DATA 189,0,202,205,185,3,208,23,189,0,
   #1204
140 DATA 203,205,202,3,208,15,201,0,240,
   11,#1288
145 DATA 56,233,1,157,0,203,169,33,141,4,
   #997
150 DATA 212,232,208,222,96,174,4,220,32,
   79,#1479
155 DATA 193,134,2,189,0,201,170,189,192,
   3,#1273
160 DATA 166,2,160,0,17,253,145,253,254,0,
   #1250
165 DATA 203,189,0,203,201,199,208,3,76,
   24,#1306
170 DATA 229,238,201,3,173,200,3,205,201,
   3,#1456
175 DATA 208,209,76,31,192,189,0,203,168,
   185,#1461
180 DATA 0,199,133,253,185,0,200,133,254,
   189,#1546
185 DATA 0,202,24,101,253,133,253,165,254,
   105,#1490
190 DATA 0,133,254,96,169,32,133,252,169,
   0,#1238
195 DATA 133,251,160,0,169,0,145,251,200,
   208,#1517
200 DATA 251,24,165,252,201,63,240,4,230,
   252,#1682
205 DATA 208,236,162,0,169,0,157,0,64,232,
   #1228
210 DATA 224,63,208,248,162,0,169,3,157,0,
   #1234
215 DATA 4,157,0,5,157,0,6,157,232,6,#724
220 DATA 232,208,241,169,29,141,24,208,
   169,59,#1480
225 DATA 141,17,208,96,0,0,0,0,0,0,#462
230 PRINT "■■>■■POKING DATA
   ■■": FOR Z=0 TO 43:T=0:C=0
235 FOR ZZ=0 TO 9:READ X$:X=VAL(X$):
   C=C+1:T=T+X
240 PRINT "■LINE"PEEK(63)+PEEK(64)*
   256,"ITEM"C,"TOTAL"T:POKE 49152+Z*
   10+ZZ,X
245 NEXT ZZ:READ X$:IF LEFT$(X$,1)<>
   "#" THEN PRINT "■ITEM
   MISSING!":END
250 IF VAL(RIGHT$(X$,LEN(X$)-1))<>T
   THEN PRINT "■DATA ERROR!":END
255 PRINT:NEXT Z:PRINT ">■■DATA
   OK■, POKING MORE NUMBERS...":
   CLR:X=8224
260 FOR Z=1 TO 8:GOSUB 290:
   X=X+1:N=N+1:NEXT Z
270 X=X+312:IF N<256 THEN 260
280 GOTO 300
290 Z1=INT(X/256):Z2=X-Z1*256:POKE
   50944+N,Z2:POKE 51200+N,Z1:
   RETURN
300 FOR ZZ=0 TO 31:FOR Z=0 TO 7:POKE
   51712+ZZ*8+Z,ZZ*8
310 POKE 51456+ZZ*8+Z,Z:NEXT Z,ZZ
1010 POKE 53280,6:PRINT "■■■■"
   TAB(8)"ENTER DIFFICULTY(1-9)?"
1020 GET A$:IF A$<"1" OR A$>"9"
   THEN 1020
1030 PRINT"■";:LV=VAL(A$)
1040 SYS 49152:FORZ=0TO23:PRINT"VV.A
   ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
   ■■■■■■■A.VV";:NEXTZ
1050 POKE 960,128:POKE 961,64:POKE 962,
   32:POKE 963,16:POKE 964,8
1060 POKE 965,4:POKE 966,2:POKE 967,1:
   POKE 968,LV*3-2
1070 S=54272:POKE S+5,7:POKE S+6,12:
   POKE S+24,5:POKE 970,179
1080 POKE 950,163:POKE 951,59:POKE 954,4:
   POKE 953,16
1090 TI$="000000":SYS 49183:SC=VAL
   (TI$)*5:POKE 53280,15
1100 K$=LEFT$("□YOU SCORED"+STR$
   (SC)+"□AT LEVEL"+STR$(LV)+"□
   □□□□□□□□□□□□□□□□□",36)
1110 PRINT "■■■■■■"K$"■":IF
   SC>Z(LV) THEN Z(LV)=SC
1120 FOR Z=1 TO 9:PRINT
   "■■■■■LEVEL"
   Z,"■SCORE",Z(Z)"■":NEXT Z
1130 PRINT "■π■■□PRESS S TO
   START GAME OR E TO EXIT□"
1140 GET A$:IF A$="S" THEN 1010
1150 IF A$<>"E" THEN 1140
1160 SYS 58648
```

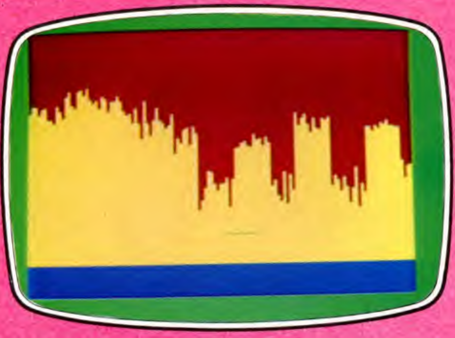
The first program section (Lines 10–310) contains the machine code DATA and the second (Lines 1010–1160) is the BASIC program.

The machine code section is contained in DATA statements and there are checksums in case you make any errors in copying the lengthy list of numbers. This part of the program controls the roller. It moves it around under keyboard control on the [C=], [SHIFT], /, and ; keys and checks for any collisions with the paint. It also plots the 256 drips.

Lines 1010–1030 enter the difficulty factor. You are offered a choice of a degree of difficulty ranging from 1 to 9. Line 1040 calls the routine to set up the Y coordinates and then prints borders down each side of the screen. Line 1090 sets the clock, executes the machine code, works out the score and changes the border colour.

```
10 MODE4
20 ?&72=80:?&71=80
30 VDU 19,1,1,0,0,0
40 DIM MC□1000,YPOS□144
50 PROCASS
60 CLS:FOR T=YPOS TO YPOS+144:
   ?T=255
70 NEXT
80 PRINTTAB(10,10)"1-VERY EASY"TAB
   (10,12)"2-EASY"TAB(10,14)
   "3-NORMAL"TAB(10,16)
   "4-DIFFICULT"TAB(10,18)
   "5-IMPOSSIBLE"
```

```
90 PRINTTAB(10,22)"WHICH ONE";
100 A=GET-48:IF A>0 AND A<6 THEN
   ?&77=A*20 ELSE 100
110 CLS:VDU5:MOVE0,76:DRAW1280,76
120 TIME=0:CALL MC
130 VDU4:PRINTTAB(7,29)"YOU SCORED ";
   TIME;" POINTS"TAB(14,30)"ON LEVEL ";
   ?&77/20
140 *FX15,0
150 END
160 DEF PROCASS:FOR T=0 TO 2 STEP 2
170 P%=MC:[OPT T
180 .DRIP:LDA &FE64:AND #96:BNE D3:
   LDA &75:CLC:ADC #16:TAX
190 LDY YPOS,X:DEY:CPY #20:BCS D2:RTS
200 .D2:TYA:STA YPOS,X:JSR PLT
210 .D3:DEC &75:BPL D4:LDA #127:STA&75
220 .D4:DEC &76:BPL DRIP:LDA &77:STA
   &76:JSR BRUSH:JMP DRIP
230 .PLT:LDA #25:JSR &FFEE:LDA #69:
   JSR &FFEE:JMP XY
240 .MVE:LDA #25:JSR &FFEE:LDA #4:
   JSR &FFEE:JMP XY
250 .BAR:TXA:CLC:ADC #24:TAX:LDA #25:
   JSR &FFEE:LDA #5:JSR &FFEE
260 .XY:LDA #0:STA &70:TXA:ASL A:ROL
   &70:ASL A:ROL &70
270 ASL A:ROL &70:JSR &FFEE:LDA &70:
   JSR &FFEE:LDA #0:STA &70:TYA
280 ASL A:ROL &70:ASL A:ROL &70:JSR
   &FFEE:LDA &70:JSR &FFEE:RTS
290 .BRUSH:LDA #18:JSR &FFEE:LDA #0:
   JSR &FFEE:JSR &FFEE
300 LDX &71:LDY &72:STY &74:INC &74:
   JSR MVE:JSR BAR:LDY #24
310 .B1:LDA YPOS,X:CMP &74:BEQ B13:
   CMP &72:BNE B15
320 .B13:ADC #3:STA YPOS,X
330 .B15:DEX:DEY:BPL B1
340 .B2:LDA &71:STA &73:LDA &72:STA &74:
   LDA &EC:CMP #128+97:BNE B3:DEC
   &73:JMP B7
350 .B3:CMP #66+128:BNE B4:INC &73:
   JMP B7
360 .B4:CMP #55+128:BNE B5:INC &74:
   JMP B6
370 .B5:CMP #86+128:BNE B8:DEC &74
380 .B6:LDA &74:CMP #20:BCC B8:CMP
   #240:BCS B8:STA &72:JMP B8
390 .B7: LDA &73:CMP #15:BCC B8:CMP
   #120:BCS B8:STA &71
400 .B8:LDA #18:JSR &FFEE:LDA #0:JSR
   &FFEE:LDA #1
410 JSR &FFEE:LDX &71:LDY &72:JSR MVE:
   JSR BAR
420 .B9:RTS
430 .CH:LDY #24
440 .C1:LDA YPOS,X
450 ]NEXT:ENDPROC
```



Paint runs on the Dragon

The assembly language from Line 160 on controls the movement of the roller via the Z, X, P and L keys. It also controls the drips and the score.

Line 20 sets up the starting point of the roller. Lines 30–50 define the colour of the paint, set aside memory and assemble the machine code. Lines 60–70 set up the starting point of the 128 drips. Lines 80–100 contain the menu for the degree of difficulty and set up the delay factor which occurs each time the paint is pushed up the screen. Lines 110–120 play the game and Line 10 shows the score.

For the Tandy change **223** to 247 and **2654** to 2678 in Line 1070.

```
10 CLEAR200,15799:CLS
20 FORK=0TO13:T=0:FORJ=0TO25:READ
   A:T=T+A
30 POKE15800+K*26+J,A
40 NEXT:READA:IF T<>A THEN PRINT
   " ERROR IN DATA IN LINE";
   1000+K*10:END ELSENEXT
50 CLS:PRINT@6,"SELECT SKILL LEVEL"
60 PRINT@200,"1 - EASY":PRINT@232,
   "2 - SIMPLE":PRINT@264,
   "3 - MIDDLING":PRINT@296,
   "4 - DIFFICULT":PRINT@328,
   "5 - IMPOSSIBLE"
70 A$=INKEY$:IF A$<"1" OR A$>"5"
   THEN 70
80 LV=VAL(A$):POKE16162,6-LV:POKE
   16164,128+64*(LV>2):POKE16167,RND
   (256)-1
90 PMODE3,1:PCLS2:SCREEN1,0
100 COLOR4:LINE(0,0)-(255,0),PSET:
   COLOR3:LINE(0,168)-(255,191),PSET,BF
110 DEFUSR0=15800:S=USR0(0)
120 SC=0:CLS:FORK=5TO0 STEP-1:
   SC=SC*256+PEEK(16173+K):NEXT
130 PRINT@8,"YOU SCORED ";SC
140 PRINT@161,"PRESS ANY KEY FOR
   ANOTHER GO":A$=INKEY$
150 IF INKEY$="" THEN150 ELSE50
1000 DATA 127,63,33,127,63,37,79,95,253,
   63,45,253,63,47,253,63,49,142,19,14,191,
   63,51,158,186,48,2585
1010 DATA 137,1,0,191,63,41,48,137,19,223,
   191,63,43,204,0,128,142,63,53,167,128,
   90,38,251,141,29,2591
1020 DATA 182,63,37,176,63,36,183,63,37,38,
   243,141,119,23,0,145,190,63,34,48,31,38,
   252;125,63,33,2426
1030 DATA 39,226,57,206,63,53,141,59,196,
   127,52,4,51,197,166,196,198,32,61,211,
   186,31,1,53,4,31,2641
1040 DATA 152,84,84,58,230,132,38,1,57,
   132,3,64,139,3,198,3,74,43,4,88,88,32,249,
   234,132,231,2553
1050 DATA 132,108,196,166,196,129,168,37,
   5,134,1,183,63,33,57,190,63,38,79,95,179,
   63,38,36,2,48,2439
1060 DATA 31,179,63,38,36,2,48,31,195,255,
   254,36,2,48,1,52,16,163,225,37,3,131,0,1,
   253,63,2163
1070 DATA 38,57,134,247,127,63,40,120,63,
   40,183,255,2,246,255,0,193,223,38,3,124,
   63,40,26,1,73,2654
1080 DATA 129,127,34,233,57,190,63,51,198,
   3,134,85,167,128,90,38,251,190,63,51,
   116,63,40,36,10,31,2578
1090 DATA 16,203,3,196,31,39,2,48,1,116,63,
   40,36,8,31,16,196,31,39,2,48,31,116,63,
   40,36,1451
1100 DATA 8,188,63,43,34,3,48,136,32,116,
   63,40,36,14,188,63,41,37,9,48,136,224,52,
   16,141,13,1792
1110 DATA 53,16,191,63,51,198,3,111,128,90,
   38,251,57,31,16,142,63,53,147,186,52,4,
   196,31,88,88,2347
1120 DATA 48,133,53,4,88,73,88,73,31,88,
   73,137,92,134,12,225,132,38,4,106,132,
   141,6,48,1,74,2034
1130 DATA 38,243,57,52,22,142,63,45,198,6,
   26,1,166,132,137,0,167,128,90,38,247,
   53,150,1,3,0,2205
```

The Dragon/Tandy machine code routine handles the score and drips. It then checks whether you are pressing the arrow keys, and if one is depressed the roller is moved accordingly. If a drip is moved up, it executes a delay and increases the score.

In the basic program, Line 10 makes space for the machine code and calls the routine to read in the machine code from data statements. Lines 20–40 read in the machine code from the data statements. Lines 50–80 set the difficulty level and poke the delay variable accordingly. Line 100 sets the screen and calls the initialisation machine code routine and then the game loop machine code.

Lines 1000–1130 contain machine code.

LOGO: BEYOND THE DRAWING BOARD

- CREATING SPRITES
- ANIMATION
- SPEED CONTROL
- MATHEMATICS
- WORDS

Move on from line drawings to animated cartoons. But there's more to LOGO than just pictures, as you will see from its maths and word-handling abilities

This, the third and final part of the series on LOGO looks at an exciting aspect of Turtle graphics which appears in some versions of the language—sprites. These are multiple Turtles with redefinable shapes which can be set in motion and used to create simple animations. As with the sprites which are available in BASIC on the Commodore 64, they can be controlled and defined easily. At least three of the four versions of LOGO for the BBC and Electron have the facility to incorporate sprites, but, because of limited memory, you will need to purchase a 'sprite board' to fit into the computer. Commodore LOGO has sprites available on a utilities disk, supplied with the LOGO disk.

Sprites enable the LOGO user to step beyond drawing pictures and create simple animated cartoons. In particular, children find sprites exciting and are often motivated to learn other programming skills through wanting to control them. Sprites have the same qualities as Turtles—in fact, the screen Turtle is itself a pre-defined sprite. In the same way as with the Turtle, your sprite can move, turn, raise and lower its pen and hide. Different versions of LOGO have different numbers of sprites, and each sprite has its own number from ∅ upwards. The Turtle



is sprite Ø. When you type in a normal Turtle graphics command you are talking to sprite Ø, and the Turtle responds.

LOOKING AT SPRITES

All the machines apart from the Commodore load the sprites together with LOGO. To use sprites with Commodore LOGO you must first load them from the utilities disk which is supplied with the LOGO disk. Insert the utilities disk in the disk drive and type:

```
READ "SPRITES
```

After loading the sprites you speak to them with TELL followed by the number of the sprite or sprites you wish to address. Sprites other than sprite Ø are invisible, and sit in the centre of the screen waiting for a command. In Commodore LOGO, if you want to see the sprite, you must type ST before giving its first command. In other versions the sprites become visible automatically when you speak to them for the first time, after that you must use ST.

To send sprite 1 forward 50 units type:

```
TELL 1 FD 5Ø
```

If you then type RT 9Ø sprite 1 will turn right 90 degrees. Sprite Ø will stay in the same position.

To turn sprite Ø 90 degrees to the right type:

```
TELL Ø RT 9Ø
```

If you then type: FD 5Ø sprite Ø will move forwards 50 units and sprite 1 will stay still. The other sprites are invisible in the HOME position.

```
TELL 1 HOME
TELL Ø HOME
```

This will bring them back to the centre of the screen.

```
TELL Ø FD 5Ø
TELL 1 RT 9Ø FD 5Ø
TELL 2 RT 18Ø FD 5Ø
TELL 3 LT 9Ø FD 5Ø
```

This will send four sprites out in a cross.

In some versions of LOGO you can talk to more than one sprite simultaneously by using square brackets, as you can see if you type:

```
TELL [Ø 1 2 3]
RT 135
FD 1ØØ
HOME
```

to make an interesting pattern. This is not possible in Commodore LOGO which only allows communication with one of the sprites at any one time.

Sprites which can be spoken to simultaneously can carry out procedures simultaneously. For example:

```
TO STAR :SIDE
REPEAT 5 [FD :SIDE RT 144]
END

TO POSSPRI
TELL 1 RT 72
TELL 2 LT 72
TELL 3 RT 144
TELL 4 LT 144
END

TO SPRISTAR :SIDE
POSSPRI
TELL [Ø 1 2 3 4] ST
FORWARD :SIDE
STAR :SIDE
END
```

STAR defines a five pointed star requiring an input called SIDE.

POSSPRI positions five sprites in different directions.

SPRISTAR makes the sprites visible and tells each sprite to move forwards distance SIDE and draw a star size SIDE.

TELL [Ø 1 2 3 4] opens communication with sprites Ø, 1, 2, 3, and 4 simultaneously. It's like a party political broadcast, sent out on all channels at once, so there's no avoiding it—but remember that Commodore LOGO only allows you to speak to one sprite at a time.

CREATING SPRITES

LOGO allows you to create your own shapes for the sprites. To do this you must enter the sprite editor. To use the sprite editor in Commodore LOGO you must insert the utilities disk and type:

```
READ "SPRED
```

The file SPRED also contains the sprite commands in the file SPRITES so loading SPRED will give you the commands to use sprites and the sprite editor.

The sprite editor contains a number of screens upon which you can plan out shapes for the sprites. You can only draw on one screen at a time. The screens, and hence the shapes you draw on them, have numbers. In Commodore LOGO you edit the shape of the sprite you are talking to. In other versions you edit a particular screen.

If you are talking to sprite 1 on Commodore LOGO, EDSH will put you into the sprite editor for the shape being used by sprite number 1. This will normally be shape No 1. EDSH stands for EDit SHape.

In other versions of LOGO, EDSH requires an input. EDSH 1 allows you to edit the sprite shape on screen No 1. It makes no difference which sprite you are currently addressing, or which shape the sprite has.

When you type EDSH a grid appears on the screen. This is the sprite editor, and in Commodore LOGO it contains the image of the current sprite on a 24 × 21 grid—the other versions are blank. Each character position on the grid represents a pixel of the sprite, and each can be filled in or blanked out causing the corresponding pixel on the sprite to be filled or emptied.

The editing cursor is moved right, left, up or down with the cursor keys. The character marked by the cursor can be filled if empty, or emptied if filled, by pressing special keys. Merely passing the cursor over a character will not change it. In Commodore LOGO the * and + key both fill the character beneath the cursor. The + key leaves the cursor in the same place and the * moves it on to the next character. The — key deletes the character beneath the cursor. The [DEL] key deletes the character beneath the cursor and moves the cursor one space to the left. Holding down [SHIFT] and pressing [CLR] blanks out every character in the editor and creates an 'invisible sprite', or blank grid.



It is a good idea to plan out your sprite shapes on a piece of graph paper before using the sprite editor. When you are satisfied with the shape on screen, you exit from the sprite editor with ESC or CTRL and C.

Once you have created different sprite shapes you can assign them to the sprites with the command SETSHAPE. In Commodore LOGO the initial shape numbers will correspond to the sprite numbers. Sprite No 1 will have shape number 1 and so on. In other versions of LOGO all the sprites have shape Ø which can only be changed with SETSHAPE. For example:

```
TELL 1 SETSHAPE 1
```

will give sprite No 1 the shape which exists in screen No 1 in the sprite editor.

```
TELL 1 SETSHAPE 2
```

will give sprite No 1 the shape that exists on screen No 2.

In Commodore LOGO:

```
TELL 1 SETSHAPE 2
```

will give sprite number 1 the same shape as sprite number 2. Changing the shape of sprite No 2 with EDSH while talking to sprite 2 will also change the shape of sprite 1. In case you feel like someone holding four phone conversations at the same time, and forget who you're talking to, the command WHO will tell you which sprite you're addressing. Type:

```
TELL 2
WHO
```

and LOGO will respond:

```
RESULT: 2
```

ANIMATION

Once you have mastered the sprite editor it is a small step to creating simple animated graphics. To get a horse to gallop across the screen we could use EDSH 1 to draw the horse in one position and EDSH 2 to draw it in another.

```
TO CLIP
TELL 1 SETSHAPE 1
REPEAT 5 [FORWARD 1]
END

TO CLOP
TELL 1 SETSHAPE 2
REPEAT 5 [FORWARD 1]
END

TO GALLOP :FURLONG
REPEAT :FURLONG [CLIP CLOP]
END
```

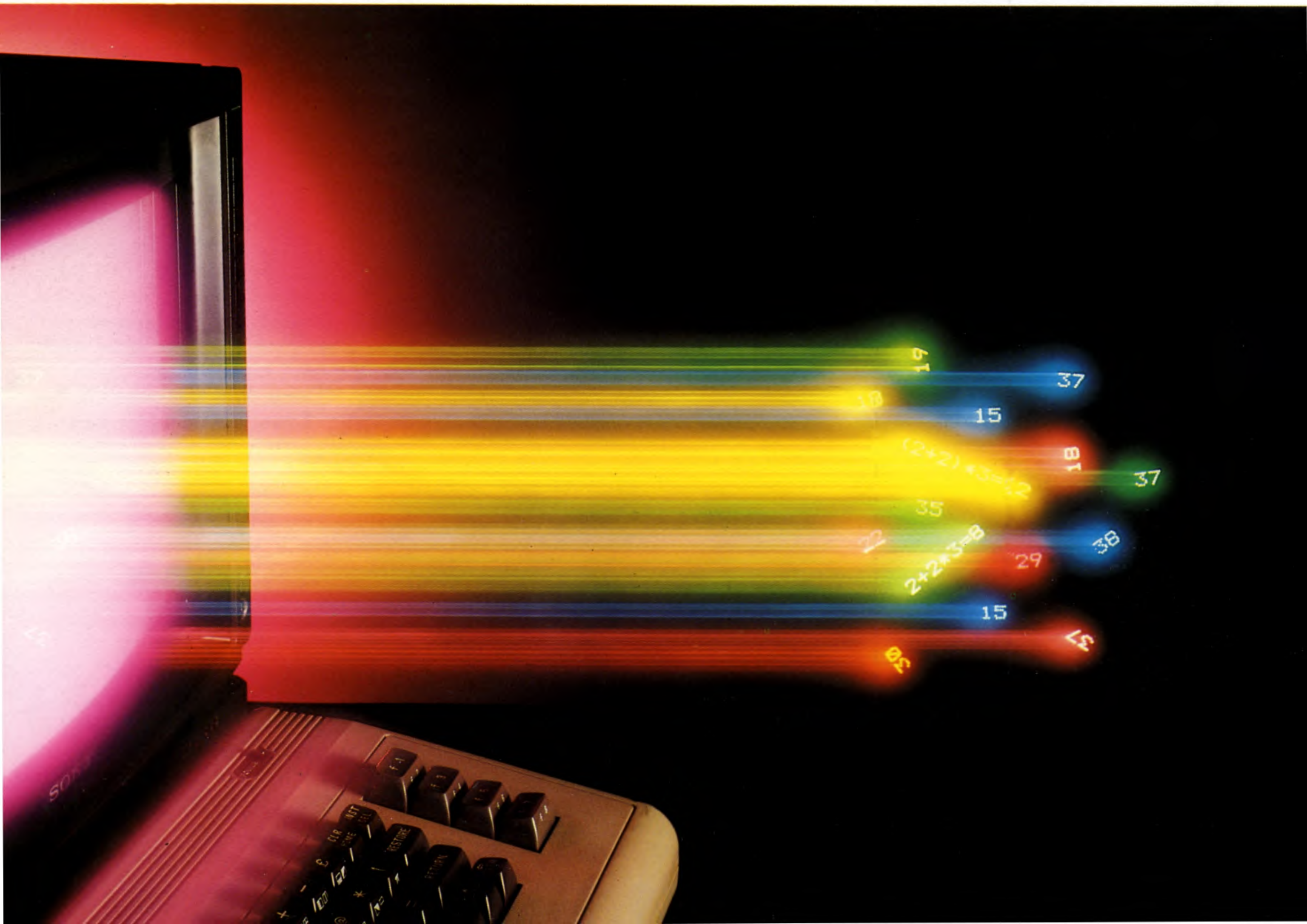
Using REPEAT 5 [FORWARD 1] instead of FORWARD 5 gives LOGO more to do and makes the sprite move more slowly. CLIP gives the sprite the first horse shape and moves it forwards five units. CLOP changes the shape to the second position of the horse and moves it forward another five units.

GALLOP requires a numerical input for the value of FURLONG which will tell the horse how many times to CLIP and CLOP.

GALLOP 2Ø will make the horse CLIP CLOP 20 times and move forwards 200 units. Alternating between CLIP and CLOP will swap the sprite between shape 1 and shape 2 and give the effect of a galloping horse. The movement could be made more sophisticated by adding extra shapes for the sprite to assume during its journey.

SPEED CONTROL

Some versions of LOGO have a command



called SETSP for SETSPeed. SETSP requires an input and sets a sprite in motion at the speed determined by the input.

```
TELL 1 SETSP 100
```

will set sprite number 1 in motion at speed 100. You can then address another sprite and sprite 1 will remain in motion until it is told otherwise. Its constant state will be speed 100.

Commodore LOGO does not have SETSP. It can be simulated with:

```
TO MOTION :SPEED
FORWARD :SPEED
MOTION :SPEED
END
```

This is a simple recursive procedure which will set a sprite in continual motion. You will not be able to give another command until you have interrupted MOTION with CTRL and the G key.

To create a simple 'cartoon' you can draw the background with the turtle, design your 'animation shapes' with the sprite editor, and use SETSH to create moving people, objects or machines.

MORE THAN MEETS THE EYE

Because of the popularity of Turtle Graphics and exciting features such as sprites, LOGO is often mistaken as a 'graphics only language', whereas in fact it has full mathematics, and word and list processing capabilities. LOGO's ancestor, LISP, was designed to handle words and lists in the world of Artificial Intelligence. So the next sections look at some of LOGO's word and list processing features and some of its mathematical capabilities.

MATHEMATICS

LOGO does not distinguish between integers (whole numbers such as 1, 56 and 1984) and real numbers (numbers with a decimal fraction such as 1.2, 34.001 and 6.345789).

Mathematical operators are the same as in BASIC. You add and subtract with + and −, multiply with * (not with x) and divide with /. All mathematical operations must be written on the same line, not in columns as with pencil and paper.

If you tell the Turtle to move

```
FORWARD 33+33
```

LOGO will do the sum and move the Turtle 66 units forwards.

```
BACK 120/3
```

will move the Turtle back 40 units.

Typing:

```
FORWARD 6−3*2
```

could lead to confusion. In BASIC, the rules of priority mean that the multiplication would be done before the division, but does LOGO subtract 3 from 6, multiply the answer by 2, and move the Turtle 6 units forwards; or does it multiply 3 by 2 first, and subtract that from 6, getting 0 and leaving the Turtle where it is? In fact, the same rules apply as in BASIC. LOGO does anything that is enclosed in brackets first, it multiplies and divides before it adds and subtracts, and, obeying the above rules, it works from left to right.

You can see the results of your calculation using PRINT, which requires an input, and prints it on the screen.

PRINT 2+2*3 gives 8 on the screen.
PRINT (2+2)*3 gives 12.

You can also use PRINT in a procedure:

```
TO CUBE :NUMBER
PRINT :NUMBER*:NUMBER*:NUMBER
END
```

CUBE requires a numeric input and gives you the cube of that number. For example:

```
CUBE 3
27
CUBE 10
1000
```

This is a procedure which gives the average of two numbers:

```
TO AVERAGE :FIRST :SECOND
PRINT (:FIRST+:SECOND)/2
END
```

Because FIRST and SECOND are in brackets, LOGO adds them before dividing by 2. Various inputs will show you how this works. The procedure accepts two inputs, like:

```
AVERAGE 2 4
```

to give a result of 3 or:

```
AVERAGE 1 2
```

which results in 1.5 being printed up. You might like to try extending the program to take more inputs.

Note the way in which the calculations above are entered. The mathematics we are used to has the arithmetical sign between the numbers involved, as in 2*2=4, 3+3=6. This is known as 'infix notation' which just means that the symbol is inserted between the numbers that you are computing.

In LOGO the command is usually given followed by a number of units, as in FORWARD 100. This is known as 'prefix notation'. The command precedes, or prefixes the number it is using. Some versions of LOGO use prefix as well as infix notation for mathematical calculations.

4/2 could also be written as DIVIDE 4 2
6+7 could be written as SUM 6 7
3*4 could be written as PRODUCT 3 4
8−7 could be written as SUBTRACT 8 7

When using SUBTRACT and DIVIDE, remember the first number is the one you are SUBTRACTing from or DIVIDEing into.

A WAY WITH WORDS

Using the " symbol after PRINT tells LOGO to treat the string of characters following " as a word. Here are a few examples:

```
PRINT "SWIM
```

results in SWIM being printed

```
PRINT "BORAGTHUNG
```

displays BORAGTHUNG

```
PRINT "1FORYOU2FORME
```

gives 1FORYOU2FORME and

```
PRINT "A2599*9(S3&=
```

prints up A2599*9(S3&=.

As this demonstrates, LOGO treats almost any combination of symbols as a word if they follow " although further punctuation can upset this. But if you try entering:

```
PRINT SWIM
```

LOGO will respond with I DON'T KNOW HOW TO SWIM.

This isn't because LOGO has never been in the water but because there was no " before SWIM. Without the quote marks, LOGO tries to evaluate SWIM. Because it is not a number, LOGO assumes it must be a procedure, and since there is no procedure called SWIM, LOGO gives an error message.

The following three examples show how the entry following the quote marks is analyzed:

```
PRINT "2+2
```

gives 2+2

```
PRINT 2+2
```

gives 4 and

```
PRINT "2 + 2
```

gives 4.

PRINT "2+2 tells LOGO to print the three characters 2+2. Without quotes, PRINT 2+2 tells LOGO to evaluate the expression 2+2, the result is 4. The third result is the interesting one. A space indicates the end of a word in LOGO, so the computer expects to



print the first 2 in PRINT "2 + 2. However, after the space it meets the +. Because of the space, it treats this as a normal mathematical operation and so it tries to apply it—checking back to see if the word before can be treated as a number. It can, so it continues to add it to the following number. The result is 4.

If you type:

PRINT "'MORNING DOROTHY

LOGO replies

'MORNING
I DON'T KNOW HOW TO DOROTHY

Because of the apostrophe, there is no quote before DOROTHY. LOGO does not treat it as a word and expects it to be a procedure. There is no procedure named DOROTHY so it gives the appropriate error message.

PRINT "'MORNING "DOROTHY

gives a response of

'MORNING
YOU DON'T SAY WHAT TO DO WITH DOROTHY

The quotes tell LOGO that DOROTHY is a word. DOROTHY is not immediately preceded by a command, so LOGO doesn't know what to do with the word and it prints an error message.

HANDLING LISTS

LOGO combines words into groups to form lists. A list is enclosed in square brackets. We have already encountered lists in turtle graphics, for example [FORWARD 10 RIGHT 30] and [HEXAGON] are both lists. A list can consist of commands, words or other lists. Given:

PRINT ['MORNING DOROTHY]

LOGO replies:

'MORNING DOROTHY

Trying these longer examples gives interesting results showing how the brackets are understood. Only the outer pair are recognized as indicating the extent of the list—any others are just read as part of the string.

PRINT [1 2 BUCKLE MY SHOE 3 4 KNOCK AT THE DOOR]

prints

1 2 BUCKLE MY SHOE 3 4 KNOCK AT THE DOOR

while

PRINT [HERE IS AN EXAMPLE OF A LIST [THIS LIST CONTAINS FIVE WORDS]]

displays

HERE IS AN EXAMPLE OF A LIST [THIS LIST CONTAINS FIVE WORDS]

and

PRINT [[5 6] [GOSH] [#%] [IS IT POETRY?]]

gives

[5 6] [GOSH] [#%] [IS IT POETRY?]

Many of the features of using words and lists use the same as those you encounter using Turtle Graphics. Writing procedures, recursion and using variables are common to both areas of LOGO. But there is an aspect of word and list processing which does not occur in Turtle graphics.

In Turtle graphics you are always talking to the Turtle. You do not have to specify that, when a procedure calls another procedure, such as RACETRACK calling OUTER, the second procedure is to be used with the Turtle.

However, words, lists and numbers do not know what you want to do with them and you must explain everything carefully to them; where to go and what to do.

If you had written AVERAGE as:

```
TO AVERAGE :FIRST :SECOND
(:FIRST + :SECOND) / 2
END
```

and then typed:

AVERAGE 3 4

LOGO would reply with something like:

YOU DON'T SAY WHAT TO DO WITH 3.5 IN LINE (:FIRST + :SECOND) / 2 AT LEVEL 1 OF AVERAGE.

In the original procedure you told LOGO to PRINT that line, so it knew what to do.

LOGO has a similar reaction to PRINT "'MORNING "DOROTHY, as in the example above.

There are two parts to handling numbers, words and lists. The first is the 'working out stage' where the numbers or data are manipulated, such as doing the sum in AVERAGE. The second part is 'what to do with it when it's worked out'. This might be simply printing the result on the screen using PRINT, or sending the information to another procedure.

OTHER NEW LANGUAGES

The three parts of this series on LOGO should have supplied you with enough information to allow you to experiment with this language. The next issues will be dealing with PASCAL but later on you will see some examples of LISP and FORTH.

UNDERSTANDING THE OPERATING SYSTEM

Without the Operating System, working with your computer would be hard indeed. Get to know about this fundamental facet of your micro

All computers work in, and understand, just one language—machine code. This is an interpretation of the minute variations of electrical voltage on which the computer operates. It can be represented by the user as a simple set of easily remembered letters and figures, or as difficult as a seemingly incomprehensible list of binary numbers. Whatever the form, few people can work comfortably with machine code, so you need a link or interface with the computer to enable communication at a simpler level. This interface is the Operating System (OS)—a chunk of software which manages the functions of the computer and makes it possible for you to communicate with the machine.

Whatever language you use, be it BASIC or one of the others covered in the Languages course starting on page 1264, this language still uses the operating system as its means of controlling the computer.

Although it is not necessary to understand in detail how the OS works, knowing about it helps to extend your programming skills and get more out of your micro. The extent to which you delve into the OS depends on the design of your computer. Within this limit,



- WHAT IS THE OS?
- WORKING TO ORDER
- UNDERSTANDING BASIC
- ENTRY POINTS
- MEMORY BLOCK
- REGISTER SET
- CASSETTE HANDLING
- TEXT MANAGEMENT
- PRINTER VARIABLES
- ROUTINES OF INTEREST

this article aims to acquaint you with the OS of your micro and show how you can use some of its extensive range of fast-acting routines.

WHAT IS THE OS?

The OS is just a form of sophisticated machine code program that lets the processor within a computer respond to your commands. It controls the machine's interactions with the outside world, the screen, keyboard, sound generator and other input and output ports. The OS also ensures that memory space is properly allocated and that the processor and you use it efficiently. When you switch on the computer, the OS initializes certain ROM routines and sets up the internal registers and pointers—all before it gives you the go-ahead sign that it is ready. On all the machines covered here, the operating system is normally concerned with handling your BASIC commands, but it is also able to deal with other languages.



Most of the OS's minute-to-minute business is to do with setting and checking pointers. For example, it is important that the processor knows the start and end of BASIC programs. As the program is edited, the size changes and the pointers must be updated.

The same principle applies to the setting up and maintenance of storage space for variables and, especially, arrays. When an array is DIMensioned, an area of memory has to be reserved for it and then cleared after the array is no longer needed. It is essential that the OS is efficient in the way it manages memory, otherwise a few large programs or arrays would use up all the memory. This management of memory is called *housekeeping*, and good housekeeping is one of the hallmarks of an efficient computer.

The OS incorporates a series of software routines, and these have been carefully arranged so that they can be called by the programmer, as well as by the OS itself. Most of the routines are not needed by the BASIC programmer, because the functions they perform are embedded into commands you would normally enter from BASIC. A good example of this is the BASIC keyword INPUT. This actually uses several OS routines, including input, channel setting, keyboard scan and buffer transfer, but the BASIC user can call them all with the one command.

Another example of an OS routine being called from BASIC is the Commodore statement CLR. This forces the OS to do any housekeeping as necessary. The effect is to make any RAM that has been used (but is no longer needed) available to you, the programmer, by wiping all variables.

WORKING TO ORDER

Whenever you give the computer a command, the OS brings the appropriate section of the computer into action. For example, if you type an A at the keyboard, it is the OS that instructs the micro to output A on the screen. To do this, it detects that a key has been pressed. On some machines, including the Spectrum and Commodores, the OS polls the keyboard regularly to detect keypresses. On others, such as the Acorns, a keypress sends an interrupt signal to the OS, which scans the keyboard to see which key was pressed.

To react to a keypress, the OS jumps to the routine it uses to output a character. This is one of many machine code routines which are executed rapidly. If you now press RETURN or ENTER, the OS runs a 'new line' routine.

UNDERSTANDING BASIC

When you RUN programs in a language, such as BASIC, the instructions held in memory are interpreted, or translated into the machine code the computer understands. At the same time, the code sets up the registers so that the OS routines can be accessed. In this way, the same routines that let the computer respond to your input at the keyboard, say, are used by the OS to execute your BASIC programs.

Why then, you might ask, is BASIC so much slower than machine code, since it makes use of these fast-acting routines? The answer is that the language must be interpreted from keywords—such as PRINT—and other symbols before the OS routines are accessed. The long time needed to translate BASIC is the reason many arcade games, and any programs that must respond fast, are written in machine code.

The time it takes to write in machine code is generally reckoned to be about ten times the time you would spend on a BASIC program to do the same task, but the result may be fifty times faster. However, the ease of program writing is why most people stick to BASIC, unless machine code is absolutely necessary.

It would still be greatly to the advantage of BASIC programmers to be able to access OS routines directly, rather than via the interpreter, and take advantage of the many efficient

routines. Unfortunately, the OS of most home micros is not intended to be accessed directly from BASIC.

On the Dragon, for example, the OS forms an integral part of the 16K Microsoft BASIC, and on the Commodores, the OS comprises the BASIC Interpreter, a collection of Kernal routines and a Kernal Control program. Even more restricting, the Spectrum does not allow you, except by machine code, to change the contents of the registers, which is essential for routing the OS to the various routines. The Acorns are in a category of their own. Their OS is accessed regularly from BASIC—as *FX calls and VDU commands.

Despite such a diversity of access provided on various machines, there are a number of features which users of even the most restrictive system can put to good effect, either to shorten coding or to solve the problems of bugs.

The OS routines are accessed by the keyword USR which you also use to execute a machine code routine which you have placed somewhere in memory. Try the direct command RANDOMIZE USR 0. This executes a full system reset. For BASIC programs, there are a few useful system variables that can be POKEd.

POKE 23561 with a number in the range 1 to 255 sets the time delay before the keys auto-repeat. The normal setting is POKE 23561,35. Similarly, POKE 23562,5 is the normal auto-repeat interval, but you can vary it between 1 and 255. These changes can be useful in games programming, or any program that requires user response at the keyboard.

Locations 23606 and 23607 hold the address of the start of the table of dot patterns for the character set. If you POKE 23606,8 (the low byte), the pointer is set one character along in the table, so anything you type will appear as the next character. Try this POKE, then type 1, 2, 3, 4. Notice that what you get is 2, 3, 4, 5, which gives a simple method of coding BASIC code, to discourage anyone from reading it. A program coded this way would execute normally, but output to the screen would be jumbled. If you now POKE 23607,0 (the high byte), the pointer is set to look at the start of ROM, so the characters appear as a meaningless jumble.

Location 23658 provides the only means of forcing CAPS LOCK when a BASIC program is running. POKE 23658,0 changes mode from upper to lower case characters, and POKE 23658,8 gives upper case only.

PLOT on the Spectrum specifies an absolute position from the origin of the screen, but DRAW is relative to the last point specified. Sometimes an absolute DRAW is required, and this is provided by locations 23677 and 23678. For example, enter PLOT 128,85 to place a dot at the centre of the screen. Now suppose you wish to draw a line from this point to the top right of the screen given absolute coordinate (255,175). DRAW 255,175 would, in fact, specify a point off screen but DRAW 255 – PEEK 23677, 175 – PEEK 23678 will give the desired line. By subtracting PEEK 23677 from the X coordinate and PEEK 23678 from the Y coordinate, you are returning the relative DRAW coordinates.

The nearest thing to a user clock on the Spectrum is the frame counter for the TV screen. Three locations specify this—23672, 23673 and 23674. If they are POKEd with 0 the frame counter is set to 0. These are then incremented automatically by interrupts. Here is a simple program to use this facility as a timer or clock:

```
10 POKE 23674,0: POKE 23673,0: POKE
   23672,0
20 BORDER 0:PAPER 0:INK 6:CLS
30 DEF FN t()=INT ((65536*PEEK 23674
   +256*PEEK 23673+PEEK 23672)/50)
40 LET t=FN t()
50 LET h=INT (t/(60*60))
60 LET m=INT (t/60)
70 LET s=t-((h*60)*60)-(m*60)
80 LET t$="["+STR$ h+":"+
   STR$ m+":"+STR$ s+"]"
90 PRINT AT 1,15-((LEN t$)/2);
   "□";t$;"□"
100 GOTO 100
```

Line 10 zeros the frame counter and Line 30 defines function t to read the counter. This reading is set to t (Line 40), from which hours, minutes and seconds are derived.

To access an OS or ROM routine on the Commodore, use SYS, followed by the address in decimal.

SYS 58648 resets the screen to the normal resolution and colours. This is much simpler than a series of POKEs, which is the usual method in BASIC.

SYS 58692 (58719 for the Vic 20) clears the screen. The usual method is a print followed by an inverse heart graphic symbol, but this symbol is not available on some printers. So the ROM call has a clear advantage.

To scroll the screen up a line, enter SYS 59626 (59765 for the Vic 20). This can be much simpler than graphics symbols to move the cursor.

A cold start can be forced by SYS 64738 (64802 for the Vic 20).

There are a number of ways of interacting with the OS from BASIC, depending on the complexity of the task to be done. The first is the OS Command Line Interpreter (OSCLI). Its role is to translate simple commands into OS actions. Whenever a statement appears in a program with a * in front of it, the rest of the statement is passed to OSCLI. This means that in multi-statement lines, there can be only one * and this must begin the last statement. OSCLI has two other limitations. The first is that it knows nothing about BASIC variables, so although *MOTOR 1 will switch on the tape, A = 1 : *MOTOR A will produce an error. A full list of the commands recognized by OSCLI appears in the User Guide. The second limitation is that OSCLI controls only a limited range of functions. To progress beyond these, you must use individual machine code subroutines within the OS.

ENTRY POINTS

The OS has a limited number of well-defined entry points, so programs written using one version of the OS will work with later versions. Moreover, it allows you to use a second language processor. Three such entry points will be described—OSBYTE (address &FFF4), OSWORD (address &FFF1) and OSCLI (address &FFF7).

It is not sufficient to just call a subroutine; information must be passed to it, telling it what to do. The simplest way to do this is to use the registers of the 6502 processor, which actually executes the machine code. The three main registers are the accumulator A, and the index registers X and Y. Each can hold a single eight-bit number. So the first way to use an OS routine is to place values into the A, X and Y registers, then call the routine.

Many OS actions can be controlled in this way, using the OSBYTE call, and an easy way of accessing it, using OSCLI, has been provided. The statement *FX A, X, Y means 'load the A, X and Y registers with the values given and then execute OSBYTE'. The commas in this command may (as for other OSCLI commands) be replaced with spaces. If a value is omitted, it is taken as zero.

Sometimes this method of using OSBYTE is not convenient, and the routine must be called directly. There are two ways to do this from BASIC—the statement CALL (address) and the function USR (address), where 'address' is the routine's address in memory. In both cases, before the routine is executed, the values of the integer variables A%, X% and Y% are placed into the A, X and Y registers. On return from the routine, USR has a four-byte

value constructed from the processor status register (P) and the Y, X and A registers (in order of significance). So *FX 18 could be replaced by A%=18 : X%=Ø : Y%=Ø : CALL &FFF4, or A%=18 : X%=Ø : Y%= Ø : variable=USR (&FFF4).

The simple *FX method will not work when you wish the OS routine to return a value, for example, the number of a character on the screen. In this case, provided the information fits into the processor's registers, the USR function can be used.

Where many bytes of data must be passed between BASIC and the OS, the OSWORD call can be used. In this case the data is placed into a block of memory locations. The two-byte address of this is split between the X and Y registers, and a number telling OSWORD what to do is placed into A. The usual method of access from BASIC is then via CALL.

BASIC VARIABLES

As just mentioned, there are difficulties in using OSCLI if it is necessary for it to use BASIC variables. For instance, suppose you want to SAVE an area of memory. The obvious way is to use the command *SAVE. This expects three parameters to follow: the file name, the start address, and the finish address plus 1 (both in hex) of the memory area. If these three items are in the BASIC variables A$, S% and F%, then the obvious method (*SAVE A$ S% F%) will not work. Instead, you must construct a string S$ = "*SAVE □" + A$ + "□" + STR$ ~ S% + "□" + STR$ ~ F%. The term STR$ ~ converts a number to its string equivalent in hex. To complete the operation, S$ must be passed to OSCLI. If you have BASIC 2 (Electron and some BBCs) then this can be done by the statement OSCLI S$.

Users of BASIC 1 must pass the string S$ to OSCLI by the memory-block technique. This can be summarized in the procedure PROCOSCLI below:

```
DEF PROCOSCLI ($OS%)
X%=OS% MOD 256: Y%=OS% DIV 256
CALL &FFF7
ENDPROC
```

This method assumes that a block of memory sufficient to hold OS$ has been reserved elsewhere in the program. For instance, DIM OS% 1ØØ will reserve sufficient room for OS$s up to 100 characters long. BASIC 1 users can now pass S$ to OSCLI with the statement PROCOSCLI(S$).

So if you want to use a BASIC variable with an OSCLI command, construct a string containing the command, then pass this to OSCLI as described above.

There are many hundreds of the straight-forward *FX type of command, most of which appear in your User Guide. For instance, *FX 21Ø, 1 will switch off the sound generator (recommended if you like to play games or run music programs after midnight) and *FX 21Ø, Ø switches it on again. Next consider *FX 138, X, Y. This inserts the character with ASCII number Y into buffer number X. The keyboard buffer is number Ø. The following is an example of using this OSBYTE routine with CALL:

```
10 *FX18
20 *KEYØRUN|M
30 *KEY1A%=138:X%=Ø:Q$="LIST"+
   STR$Q%+","+STR$(Q%+100)+
   CHR$13:FORI%=1TOLENQ$:Y%=
   ASCMID$(Q$,I%,1):CALL&FFF4:NEXT:CLS:
   |M
40 *KEY2Q%=Q%+100:CLS|M
50 *KEY3Q%=Q%-100:CLS|M
60 *KEY4Q%=Q%+500:CLS|M
70 *KEY5Q%=Q%-500:CLS|M
80 *KEY6Q%=ERL-ERL□MOD100|M
90 END
```

The idea is that you LOAD and RUN this program before you start a programming session. It sets up the soft keys or function keys so that every time you press [f1], Lines Q% to Q%+1ØØ are LISTed. Keys [f2] to [f5] increment and decrement Q% and clear the screen. So to look through a program, all you have to do is to alternately press key [f1] and one of the others. If your program has ended with an error message, hitting [f6] and then [f1] will LIST the lines around the error.

The heart of the program is the definition of [f1] (Line 3Ø). The first thing this does is to set A% and X% so that later, OSBYTE with A=138 can be called. Next Q$ is constructed to be the word LIST plus the strings representing the numbers Q% and Q%+1ØØ, with a comma between. The FOR I% loop then uses ASC and MID$ to place the ASCII number of each character in Q$ into Y%. The CALL &FFF4 uses OSBYTE to place each character into the keyboard buffer. In this way, the correct LIST command is typed for you. Note that Q$ is terminated with CHR$13, which is equivalent to [RETURN] being pressed at the end of the LIST command. The *FX 18 in Line 1Ø removes any previous *KEY definitions.

FETCHING NUMBERS

The next type of call to consider are those that return a value. A useful example of this type is OSBYTE with A=135. On return from this, X contains the character at the text cursor and Y contains the MODE number. The following program is a typical application of this call:

```
10 MODE4
20 PROCSETUP
30 ?&DØ=?&DØ OR 2
40 FORH%=ØTO3Ø
50 PRINT"SCREEN SAVE TEST"
60 NEXT
70 PROCSCREENSTORE
80 CLS
90 PRINTTAB(4,12)"PRESS RETURN TO
   RESTORE SCREEN"
100 REPEAT UNTIL GET=13
110 PROCSCREENREPLACE
120 ?&DØ=?&DØ□AND 253
130 END
150 DEFFNC(X%,Y%)
160 A%=135:V%=VPOS:H%=POS
170 VDU31,X%,Y%
180 A%=(USR(&FFF4)AND&FFØØ)DIV&1ØØ
190 VDU31,H%,V%
200 =A%
220 DEFFNMODE
230 A%=135
240 =(USR(&FFF4)AND&FFØØØØ)DIV
    &1ØØØØ
260 DEFPROCSETUP
270 DIM S$(1Ø)
280 FORI%=ØTO1Ø:S$(I%)=STRING$(251,
    "*"):S$(I%)="":NEXT
290 ENDPROC
310 DEFPROCSCREENSTORE
320 RESTORE
330 FORI%=ØTO FNMODE:READ N%:NEXT
340 VDU31,Ø,Ø
350 FORI%=Ø TO N%-1
360 S$(I%DIV25Ø)=S$(I%DIV25Ø)+CHR$
    FNC(POS,VPOS):VDU9
370 NEXT
380 ENDPROC
400 DATA256Ø,128Ø,64Ø,2ØØØ,128Ø,64Ø,
    1ØØØ,1ØØØ
420 DEFPROCSCREENREPLACE
430 VDU31,Ø,Ø
440 FORI%=Ø TO N%-1
450 VDU ASC(MID$(S$(I%DIV25Ø),I%MOD
    25Ø+1,1)):NEXT
460 ENDPROC
```

The program stores an entire screen of text in a string array. The test program in Lines 1Ø to 13Ø PRINTs a message on the screen, stores the screen, clears it and then restores it after [RETURN] is pressed. To use these procedures, you must switch off the screen scrolling; Line 3Ø does this and Line 12Ø switches it on again. It is one of the few omissions in the OS that a call is not provided to do this task.

The section of particular interest is FNC(X%,Y%) Lines (15Ø to 2ØØ). This function returns the number of the character on the screen at position X%, Y%. Line 16Ø sets A% to 135 for the OSBYTE call and stores the

present position of the text cursor in H% and V%. Line 170 uses VDU31 to move the cursor to X%, Y%. Line 180 actually calls OSBYTE with USR. To extract the contents of the X register on return (the second byte of USR), the value of USR is ANDed with &FF00 and then divided by &100. The function places the cursor back to its initial position in Line 190, and in Line 200 returns the ASCII value of the character.

Another variation of this call can be found in Lines 220 to 240, where the function FNMODE returns the MODE number, again using OSBYTE with A=135. This time the value needed is returned in the Y register (the third byte of USR, hence the different AND and DIV numbers). There are three other procedures in the program. PROCSETUP (Lines 260 to 290) reserves sufficient room in a string array S$() to store any text screen. PROCSCREENSTORE (Lines 310 to 380) actually stores the screen. It does this by first placing the text cursor at the top left (Line 340) and then using FNC at each point on the screen (Lines 350 to 370). The VDU9 in Line 360 moves the cursor forwards one position each time through the loop. Notice that in Line 330, FNMODE is used to find the number of characters on the screen in a particular MODE. These numbers are stored in the DATA statement (Line 400). PROCSCREENREPLACE (Lines 420 to 460) uses a FOR loop to print the string array and so restores the screen.

MEMORY BLOCK

The final type of call includes all those that use a block of memory to pass information. The general-purpose call of this type is OSWORD. Two useful forms are A=&D, which returns the last two positions of the graphics cursor, and A=&A, which gives access to the definitions of the characters used on the screen.

Here, OSWORD with A=0 is used. This inputs a string of a specified length from the current input stream (usually the keyboard). The string entry is terminated with RETURN. DELETE and CTRL U work as they should, deleting single characters or the entire input. If more than the specified number of characters are input, the extra ones are ignored and a beep is sounded. Although the action of this call could be duplicated in BASIC, the use of OSWORD saves memory and programming. The following program is an example of how it is used.

```
10 PROCSETUP
20 MODE6
30 PRINT"INPUT YOUR WORD"
40 PRINT"(less than 15 letters please)"
50 PRINT">>>□";
60 A$=FNINLINE(15,32,122)
70 PRINT''"ANAGRAMS□□>>>□";
80 L%=LEN A$
90 FOR I%=1 TO L%
100 I1%=RND(L%):I2%=RND(L%)
110 IF I1%=I2% GOTO100
120 IF I1%>I2% IT%=I1%:
    I1%=I2%:I2%=IT%
130 A$=LEFT$(A$,I1%-1)+MID$(A$,I2%,
    1)+MID$(A$,I1%+1,I2%-I1%-1)+
    MID$(A$,I1%,1)+MID$(A$,I2%+1,
    L%-I2%)
140 NEXT
150 PRINT'TAB(15) A$;
160 I$=GET$
170 *FX21,0
180 IF I$="A" GOTO 20 ELSE 90
190 END
200 DEFPROCSETUP
210 OSWORD=&FFF1
220 DIM PBLOCK 4
230 DIM STRING 100
240 X%=PBLOCK□MOD 256
250 Y%=PBLOCK□DIV 256
260 A%=0
270 ?(PBLOCK+0)=STRING□MOD 256
280 ?(PBLOCK+1)=STRING□DIV 256
290 LTH=PBLOCK+2
300 MIN=PBLOCK+3
310 MAX=PBLOCK+4
320 ENDPROC
330 DEFFNINLINE(LLTH,LMIN,LMAX)
340 ?LTH=LLTH
350 ?MIN=LMIN
360 ?MAX=LMAX
370 CALL OSWORD
380 =$STRING
```

This program prints out anagrams of any word you INPUT. Pressing A lets you enter a new word; pressing any other key generates a new anagram.

The need often arises to input information in just the form that this OSWORD routine allows, so you can use this program to provide a data entry sub-program within other programs. This facility is provided by function FNINLINE(LLTH,LMIN,LMAX) (Lines 330 to 380), which returns the input string. Its arguments are the maximum string length (LLTH) and the minimum and maximum ASCII values of characters accepted.

REGISTER SET

Before using FNINLINE, PROCSETUP (Lines 200 to 320) must be called (as in Line 10). This arranges registers for the OSWORD call. Line 210 places the address of OSWORD into a variable of the same name. OSWORD with A=0 needs a block of five memory locations to be reserved. These are to contain the low and high bytes of the address of another block of memory (to hold the input string), the maximum length of the string, and the minimum and maximum ASCII values allowed. Line 220 reserves this block at location PBLOCK, and Line 230 reserves room for a string of 100 characters at location STRING. Lines 240 to 260 set up the processor's registers; the address PBLOCK is placed in X% and Y%, and A% is set to 0. Lines 270 and 280 place the address STRING into the first two locations in PBLOCK. Lines 290 to 310 place the address of the last three locations in PBLOCK into LTH, MIN and MAX.

After all this work, FNINLINE is simple. In Lines 340 to 360, the query indirection operator (?) is used to POKE the function's parameters in the memory locations LTH, MIN and MAX. Line 370 does the OSWORD CALL. In Line 380, the string indirection operator ($) is used to get the input string from the block STRING.

Lines 10 to 190 are an example of the use of this general-purpose input routine. The crucial part is Line 60, where A$ is set to the function FNINLINE. The values of its arguments specify an input length of 15 letters with ASCII values between 32 (space) and 122 (z). The rest of the program generates anagrams of the input. Note the use of *FX 21, 0 in Line 170, which flushes the keyboard buffer and protects the program from heavy-handed users.

When you use an OS call, you are almost programming in machine code. If anything goes wrong with a machine code program, it can erase itself, so it is advisable to SAVE your programs before you RUN them.

The OS ROM routines are accessed using the EXEC command, and, additionally, there are many useful OS variables which can be PEEKed.

CASSETTE HANDLING

All cassette routines access one of two main subroutines. These are BLKIN—at address 47422 (42763 for Tandy)—which reads a block of 255 bytes from tape, and BLKOUT—address 47513 (42996 for the Tandy)—which writes a block to tape. Of more use to the BASIC programmer are the following locations:

121—Gives the current status of cassette I/O, and takes the values 0 (closed), 1 (open input) or 2 (open output). This can be PEEKed before attempting to open a file to avoid AO (already open) errors crashing your program.

144—Used by the OS as the length of the

leader tone. If you have problems with automatic level controls on your cassette, try poking a higher value in this location.

149/15Ø—These hold the value for the delay after a MOTOR ON command is issued. POKE 149,Ø:POKE 15Ø,1 gives no delay, which is useful when using AUDIO ON/OFF with motor control.

TEXT MANAGEMENT

The next useful section of the OS is the test I/O. This includes keyboard input, screen and printer output. Frequently, a 'Press any Key' message appears in programs, with a loop such as:

```
100 IF INKEY$="" THEN 100
```

This line can be replaced with the much shorter EXEC 34Ø91 (44539 for the Tandy). If you want a flashing cursor to appear as well, use the alternative OS routine by EXEC 41194 (36Ø38 for the Tandy). Location 135 contains the ASCII code of the last key pressed, and by poking 255 in locations 337 to 345 (338 to 345 for the Tandy) before reading INKEY$, an auto-repeat can be achieved on the 32K machines. Location 329 (282 for the Tandy) is the CAPS LOCK flag. You can force lower case entry by POKEing a Ø here. A value of 255 gives upper case, and any other value gives upper case and also disables the SHIFT Ø function.

The OS routine at 47735 (433Ø4 for the Tandy) will clear the text screen.

PRINTER VARIABLES

There are a number of variables which relate to the printer that are useful to know. Location 153 can be POKEd to alter the separation between items printed separated by a comma; the default is 16. To make the POS(−2) function work correctly, location 155 should be POKEd with the printer line width (for example, 4Ø, 8Ø, 132). Location 33Ø contains the number of characters in the 'End-of-Line' (EOL) sequence. This is 1 by default. Locations 331 to 334 are the EOL characters; by default these are CR (13), LF (1Ø), Ø, Ø. These can be altered to suit a particular printer. Location 328 is the auto-linefeed flag. If set to Ø, the micro assumes the printer has automatic wrap-round (the default). Any other number causes the EOL sequence to be printed after the number of characters indicated by location 155.

Due to a bug in Dragon BASIC, the computer will hang up if you try to access the printer when one is not connected. Your program can avoid this by using (PEEK(&HFF22)AND1). This gives a value of 1 if no printer is connected, but Ø if one is connected and ready.

GRAPHICS

With regard to graphics, no useful OS ROM routines can be used from BASIC, but there are a number of locations which can be PEEKed and POKEd:

182—PMODE number of current graphics.

183/184—End address of current graphics.

186/187—Start address of graphics.

188—Start of page 1—the default is 6 but may be higher with a disk operating system.

2ØØ—Current X coordinate of graphics cursor.

2Ø2—Current Y coordinate of graphics cursor.

ROUTINES OF INTEREST

EXEC 48466 (43486 for the Tandy)—Update all joysticks—this removes the bug in the JOYSTK command.

EXEC 46ØØ4 (4Ø999 for the Tandy)—Does the same as pressing RESET.

EXEC 46Ø8Ø (41142 for the Tandy)—Restart the BASIC; gives sign-on message and NEWs program.

EXEC 36Ø55 (46481 for the Tandy)—Does a controlled Garbage Collection (or tidying up of memory) of strings. It avoids the embarrassing pauses that often occur unexpectedly in programs of this type. The amount of free string space remaining can be calculated by

```
PEEK(35)*256+PEEK(36)−PEEK(33)*256−PEEK(34)
```

By default, this is set to 200 bytes, keeping an eye on this value in programs can avoid OS (out-of-string space) ERROR occurring.



CLIFFHANGER: ROCKY II

Willie had better watch out. Those boulders are ready to roll. And in the second part of this two-part article, the rocks are going to come crashing down on him

So far, Willie is in no danger from your boulders. The rock-rolling routine is only half finished and will probably crash if you try to execute it. But he shouldn't be complacent. This second part of the routine is going to set the stones rolling. And Willie better get ready to jump out of the way.

There are two main parts to this routine, both of which were called from the routine in part one of this article. The first part prints up the boulder in its second position to make it look like it is rolling and checks to see if it has hit Willie. The second part blanks out a boulder if it has reached the edge of the screen or the surface of the water and starts it off again at the top of the screen.

Once you have keyed it in and assembled it, the boulder-moving routine should now work. Remember, though, that you must have the rest of the game in memory because other routines—like **print**—are called.

```
    org 59Ø97
bma ld hl,(57356)
    ld de,22528
    add hl,de
    ld a,(hl)
    cp 4Ø
    jr nz,bnh
    ld a,2
    ld (57336),a
bnh ld hl,(57356)
    ld a,42
    ld bc,57128
    call 58217
    inc hl
    call 58217
    ld a,Ø
    ld (57358),a
    ret
bri ld hl,(57356)
    ld bc,15616
    ld a,45
    call 58217
    ld hl,223
    ld (57356),hl
    ret
```

In the last part of Cliffhanger you had to look at the attribute of the character square underneath the boulder to see it had reached the water or was flying through the air. But here you want to look at the attribute of the position the boulder is about to move into to see if it has hit the man.

So the new boulder position is loaded into HL from the memory location it is stored in, 57,356. Remember that the variable in HL was decremented at the end of the first part of this article. So when the routine is called again, the boulder-position variable points to the screen position one character to the left of the boulder. The processor then branches to this routine and prints the next boulder picture in the same position to make it look like it is rolling. So, in fact, this boulder picture is printed in a screen position before the other one.

DE is loaded with 22,528 and this is added onto the contents of HL. The result—the pointer to the attribute of the position at which the boulder is about to be printed—is left in HL.

The HL pointer is then used to pick up the attribute of that character square and load it into A, using direct addressing. This is compared to 4Ø, which is the attribute for blue on cyan, the colour of the Willie against the sky.

If the attribute of that square is not 4Ø, and the rock has not hit Willie, the **jr nz,bnh** jumps the processor forward over the next to instructions.



TO DIE, TO SLEEP

If Willie has been mortally wounded by the oncoming rock, the variable in 57,336 has to be set to 2.

This is done by loading 2 into the accumulator and loading the contents of the accumulator into 57,336.

AND THE ROCK ROLLS ON

Whatever happens the rock rolls on one more square. If Willie is not standing there, there is no need to worry. But if he is, you must make sure that he is well and truly crushed.

HL is loaded up with the new screen position, A with 42—red on cyan—and BC with 57,128, the start address of the data for the second boulder picture (even though it is printed first). The print routine is then called, which prints up the first half of the boulder. The HL is incremented—which moves the pointer—one character square to the right—and the second half of the boulder is printed up by calling the print routine again.

You will see now that the rock moves half a character square at a time. That is why this second rock picture occupies two bytes of the data table. This half-frame jump makes the movement seem much smoother and more continuous, while the two different pictures give the impression that the rock is rolling.

- ROLLING THE ROCK
- OVERPRINT THE OLD ROCK
- PRINTING UP THE NEW ROCK
- CHECKING FOR THE EDGE
- LOOKING FOR THE SEA

The 'CLIFFHANGER' listings published in this magazine and subsequent parts bear absolutely no resemblance to, and are in no way associated with, the computer game called 'CLIFF HANGER' released for the Commodore 64 and published by New Generation Software Limited.



You will notice, too, that incrementing the rock in the routine given by the last part and printing the two sides of the boulder here back to front—from left to right—you preserve the smooth advancement of the screen print position in 57,356 and eliminate the need to check if it is still on the ground.

All that remains to be done in this part of the routine is to reset the boulder mode variable to Ø, so that the next time the routine is called the processor takes the other branch. This is done by loading the accumulator with Ø and loading it into 57,358.

ROCK 'N' RETURN

The **bri** routine is called from the routine given in part one of this article when the rock has reached the left-hand side of the screen or has fallen in the sea. All it does is to blank out the boulder and get it ready to start again.

The blanking out is done just like it was last time. The boulder position is loaded from 57,356 into HL, BC is loaded with the data for a space in ROM and A is loaded with 45 which corresponds to cyan on cyan. The **print** routine is then called which prints a cyan-coloured space over the boulder so that the character square it occupied looks just like any other piece of sky.

HL is then loaded with 223, which is the screen position of the boulder when it is at the top of the slope. This is then loaded back into 57,356 so that next time the rock-rolling routine is called the boulder starts from up there.

This routine moves the boulder four pixels to the left, then down four pixels if the slope drops away and there is space below the boulder.

```
        ORG 22784
        LDA SDØ15
        AND #2
        BEQ RET
        DEC $CØØ8
        LDA $ØØ15
        CMP #57
        BEQ MSB
YY      DEC $CØØ2
        DEC $CØØ2
        DEC $CØØ2
        DEC $CØØ2
        JSR $51ØØ
        CMP #32
        BNE CHECK
        INC $CØØ9
        INC $DØØ3
        INC $DØØ3
        INC $DØØ3
        INC $DØØ3
CHECK   LDA $CØØ8
        CMP #4
        BEQ XX
RET     RTS
XX      JSR $585Ø
        RTS
MSB     LDA #252
        STA $DØØ2
        LDA $DØ1Ø
        AND #253
        STA $DØ1Ø
        JMP YY
```

The first thing the boulder-moving routine has to do is check whether a boulder is needed. So the contents of the sprite display

enable byte in $DØ15 is loaded into the accumulator and ANDed with 2. The boulder sprite is sprite number one. ANDing the enable byte with two checks to see whether bit one is set—in other words, it checks to see whether sprite one is switched on.

If it is not, then there is no point in going on with this routine and the BEQ routine jumps to the label RET which marks the RTS instruction.

MOST SIGNIFICANT BOULDER

The next things that have to be checked are whether the boulder sprite is at the edge of 255 screen range and whether the MSB register has to be updated. This boundary occurs at 57 in our double density coordinates.

So the double density X coordinate in SCØØ8 is decremented and compared to 57. If it is 57, the processor jumps to the MSB update routine, if not it continues.

The MSB routine simply loads 252 into the X coordinate of the sprite one, which is stored in $DØØ2. The sprite is moving four pixels at a time—252 is 256 − 4. Then the contents of the MSB register at $DØ1Ø are loaded into the accumulator, ANDed with 253—that is, 255 − 2—and stored back in $DØ1Ø. This clears bit one, the most significant bit of the X coordinates.

The processor returns to the main routine after the boulder sprite X coordinate has been decremented to move it along four pixels.



LEFT, DOWN AND OUT?

The X coordinate of the boulder sprite in $DØØ2 is then decremented four times, moving it four pixels to the left. Then the routine that checks what is in the character square below it, at $51ØØ, is called.

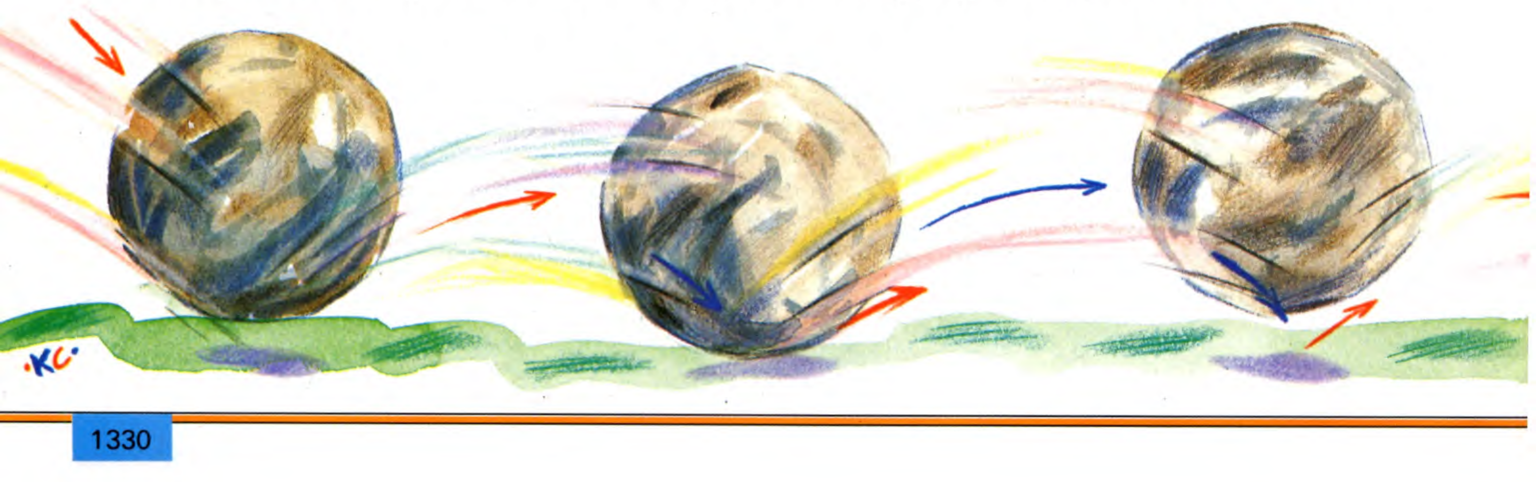
The result is returned in the accumulator. This is compared to 32—the ASCII code of a space. If there is no space the BNE instruction branches the processor on over the next little routine. But if there is a space below the boulder, the processor continues.

In that case, the double density Y coordinate in $CØØ9 is incremented to move it down the screen four pixels. And the sprite's Y coordinate in $DØØ3 is incremented four times, moving it four pixels down the screen too.

You then have to check whether the boulder has gone off the edge of the screen. So the double density X coordinate, in $CØØ8, is loaded back into the accumulator and compared with 4. It is not allowed to decrement as far as Ø otherwise it would start appearing on the other side of the screen.

If it has reached as low as 4, the processor branches over the RTS instruction and jumps to the subroutine at $585Ø which resets the variables to put the boulder back at the top of the slope.

The following routine moves the boulder down the slope. Do not forget to set up the computer as normal before you start.





```
20 FORPASS=ØTO3STEP3
30 P%=&1E1D
40 [OPTPASS
50 .Move
60 JSR&1DEE
70 LDA&88
80 LSRA
90 LSRA
100 CLC
110 ADC#3
120 CMP&79
130 BCCLb5
140 LDA#38
150 STA&78
160 LDA#46
170 STA&79
180 LDA#Ø
190 STA&75
200 STA&76
210 .Lb5
220 LDA&75
230 AND#&1F
240 BNELb1
250 INC&76
260 LDX&76
270 LDA&187C,X
280 ASLA
290 STA&75
300 .Lb1
310 LDA&75
320 AND#&80
330 BEQLb2
340 DEC&79
350 .Lb2
360 LDA&75
370 AND #&40
380 BEQLb3
390 DEC&78
400 .Lb3
410 LDA&75
420 AND#&20
430 BEQLb4
440 INC&78
450 .Lb4
460 DEC&75
470 JSR&1DEE
480 RTS
490 ]NEXT
```

BOULDER NO MORE

The first thing you have to do when you move anything on the screen is delete it in its last position—otherwise it will leave a trail of images behind it.

Here the deletion is done simply by jumping to the boulder-printing routine given in the last part of Cliffhanger. If you look back you will see that the first parameter following the machine code GCOL equivalent in that routine is a 3. This means that the logical colour which follows in the second parameter is Exclusively ORed with what is already on the screen. So if this routine is called when a boulder is on the screen, the boulder is overwritten by the background colour and disappears.

AT THE SEA SIDE?

When the boulder rolls down the slope you don't want it to run on into the sea. The height of the tide is stored in &88. But the way the sea advances the parameter here is the Y coordinate times four. So the two logical shifts right divide it by four.

Three is then added. This is to prevent any part of the boulder going into the water. The result is compared to &79, which is the current Y coordinate of the boulder.

If the boulder has not reached the sea yet, the carry flag will not be set. So the BCC instruction in Line 13Ø branches the processor over the next routine.

BACK BOULDER

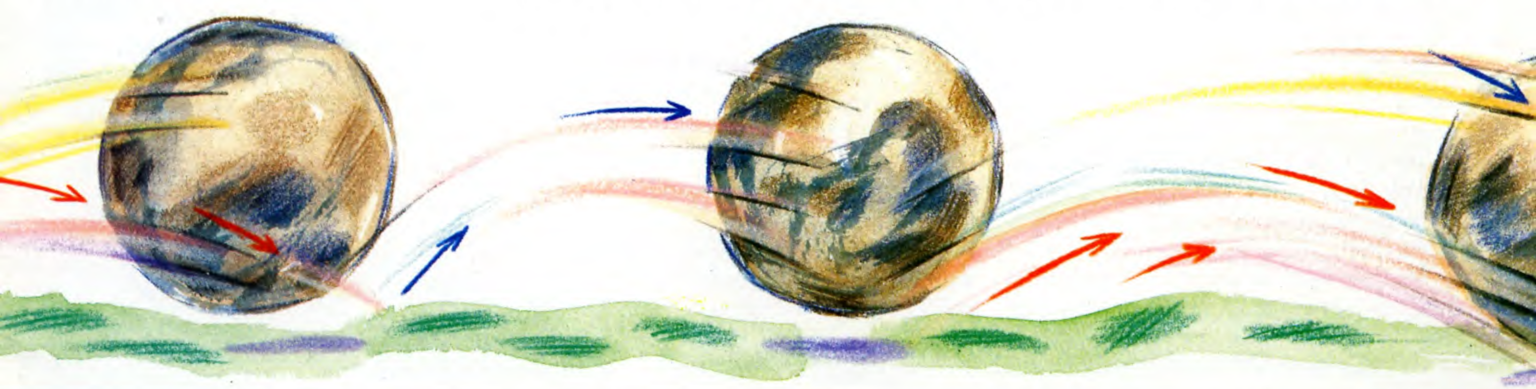
If the boulder has reached the sea though, this comparison sets the carry flag and the branch is not made and the processor goes on to the next instruction.

The X and Y coordinates in &78 and &79 are reset by storing the coordinates of the start position at top of the slope 38 and 46—in them.

The location that controls the direction the boulder is going in, &75, is reset by storing Ø in it. And the data pointer, which follows the profile of the slope, is set back to Ø in the same way.

A MOVING EXPERIENCE

The contents of &75 are loaded into the accumulator and ANDed with 1F. Although &75 has just been cleared by storing Ø in it, normally when the processor goes round this loop it contains a byte of the data table at



&187C which was given in part four of Cliffhanger on page 1040. This is the data which controls the drawing of the first slope. Using this to control the path of the boulder as well means that you don't have to worry about it falling down potholes.

The direction byte is ANDed with 1F—it is bits Ø to 4 which specify how many character squares the slope is going to continue in the same direction, remember.

The BNE instruction in Line 24Ø checks to see whether it has finished moving in any given direction. If it has the processor branches forward to Line 3ØØ. But if it hasn't the processor proceeds to the next instruction.

In that case the data pointer in &76 is incremented to move it onto the next byte of slope data. The pointer is loaded into the X register and is then used as an offset in the LDA&187C,X instruction which loads the accumulator with the next byte of the slope data.

The slope data is then shifted one place to the left. This effectively doubles it, because the boulder is going to move in half character squares—rather than the whole character squares the original slope is drawn in. The shift will not affect the direction the boulder takes—provided the right bits are picked out—only the number of places the boulder is moved in any one direction.

The direction and distance data is then saved in &75 by the STA&75 instruction in Line 29Ø.

WHICH WAY NEXT?

The direction and distance data is reloaded into the accumulator from &75. This is needed because the processor may not have come directly from the previous routine. The instruction in Line 24Ø might have branched the processor over it, direct to Line 3ØØ.

ANDing with &8Ø looks at bit seven. If it is not set and the slope is continuing flat, the processor branches over the next instruction. But if it is set and the slope is going down, the Y coordinate of the boulder is decremented.

Next, ANDing with &4Ø looks at bit six. This tells the boulder whether it is going left or not. If bit six is set the boulder's X coordinate in &78 is decremented, which moves it left. Otherwise this instruction is skipped.

Then the data is ANDed with &2Ø. Bit five tells the boulder whether it is going right. If it is set the boulder's X coordinate in &78 is incremented. If not, this is skipped.

The contents of &75 are then decremented. Effectively this only decrements the distance part of the data in the five least significant bits. If these had already been decremented to zero new data would have been picked up—there is a test in Lines 22Ø and 23Ø.

Once the distance has been decremented the processor jumps to the subroutine at &1DEE. This is the routine that prints the boulder on the screen that was given in the first part. The processor then returns.

TESTING

To test this routine you will have to have the program in memory as it calls data and routines given in earlier parts. Then key in:

```
CALL &1D77:CALL &1D9B:CALL
&1DEE:REPEAT CALL &1CCB:CALL
&1E1D:FOR A%=Ø TO 6ØØ:NEXT:UNTIL Ø
```

There are two main parts to this routine, both of which were called from the routine in part one of this article. The first part, which starts at the label BOK, decides which of the two boulder pictures to print, prints it on the screen and flips the variable which controls which boulder is printed so that the other one is printed next time. The second part, BRI, blanks out a boulder if it has reached the edge of the screen or the surface of the water and starts it off again at the top of the screen.

Once you have keyed it in and assembled it, the boulder-moving routine should now work. Remember, though, that you must have the rest of the game in memory because other routines—like CHARPR—are called.

```
ORG      19853
BOK      LDA 1826Ø
         BEQ BMN
         LDX 18253
         LDU #18Ø38
         JSR CHARPR
         CLR 1826Ø
         RTS
BMN      LDX 18253
         LDU #18Ø14
         JSR CHARPR
         LDA #1
         STA 1826Ø
         RTS
BRI      LDX 18253
         LDU #1536
         JSR CHARPR
         LDX #3Ø7Ø
         STX 18253
         RTS
CHARPR   EQU 194Ø2
```

To give the impression that the rock is rolling, there are two rock pictures in memory. These are printed on the screen alternately, so it looks like the rock is turning.

To do this the processor has to know which rock picture was printed last time. This is done by consulting a flag in 18,26Ø.

The contents of this location are loaded into the accumulator. If they are Ø, the instruction BEQ BMI jumps onto the routine that prints up one of the rock pictures. If not the processor continues to print up the other.

ROCK IT

If the processor continues, X is loaded with the contents of memory location 18,253. This is the location which stores the position the boulder is to be printed in. U is loaded with the number 18,Ø38. This is the start address of one of the rock pictures.

The processor then jumps to the CHARPR subroutine which prints the last eight bytes of the user stack in the screen position pointed to by the contents of the X register.

The CLR 1826Ø instruction changes the flag in 18,26Ø. The processor came down this branch because the flag in 18,26Ø was set. This instruction clears it. So next time this boulder routine is called, the processor will take the other branch and print the other rock picture. After the flag has been reset, the processor returns.

The next part of this routine simply prints up the other rock picture. The X register is loaded up in the same way, but the user stack pointer, U, is loaded with the start address of the other rock picture, 18,1Ø4. Then the CHARPR routine prints it up on the screen.

The processor reached this part of the routine because the flag in 18,26Ø was set to Ø. So 1 is loaded into the accumulator and stored in 18,26Ø. This flips the flag so that next time the other rock picture will be printed.

The BRI routine is called when the boulder gets to the edge of the screen or lands in the water. It blanks out the boulder and starts it off at the top of the mountain.

LDX 18253 loads up the X register with the current screen position again, and U is loaded with the start address of a piece of sky in the screen memory. Then CHARPR prints a piece of sky over the boulder.

X is then loaded with 3Ø7Ø, the start position of the boulder at the top of the slope, which is then stored in 18,253, the variable which carries the screen position where the boulder is to be printed—next time.

To test this routine, LOAD in the rest of Cliffhanger and RUN this BASIC program.

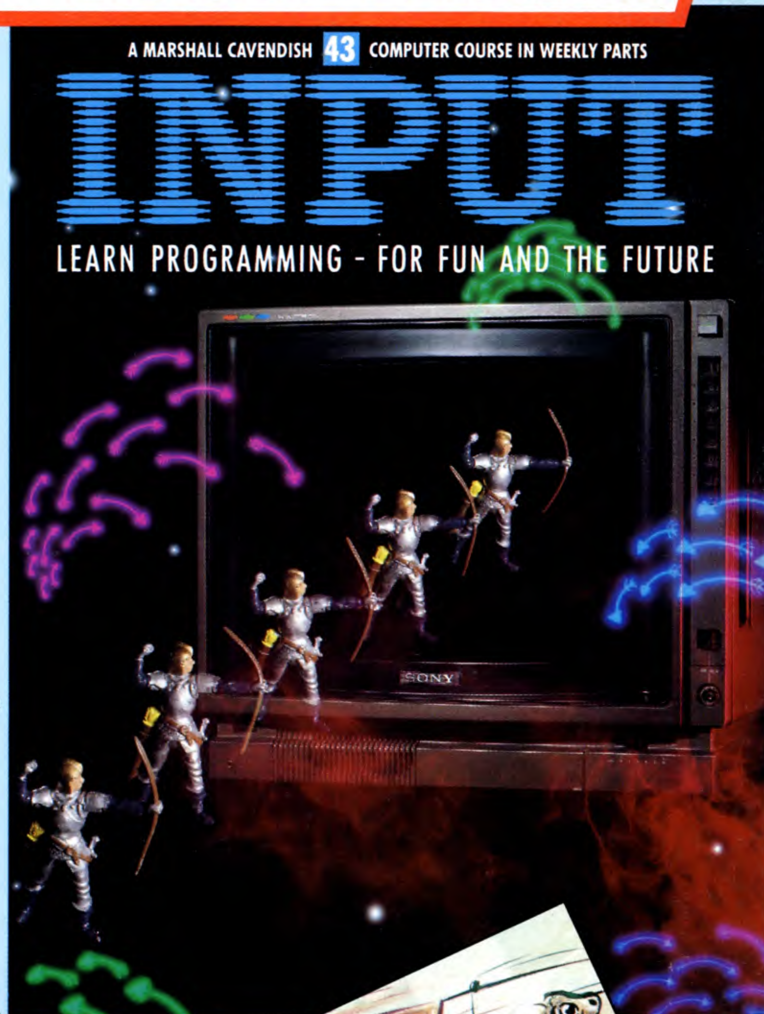
```
5 POKE 3ØØØØ,57
1Ø EXEC 19426
2Ø EXEC 19781
3Ø FOR K=1TO1ØØ:NEXT:GOTO 2Ø
```

Line 5 is required if you have SAVEd the music routine separately.

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 43...



❑ ***LANGUAGES** continues with a look at PASCAL, a language that needs careful thought before you touch the keys*

❑ *In the fourth part of **CAVENDISH FIELD**, continue to enter the programming that will enable you to fight the good fight on the battlefield*

❑ *In the arcade-type game that's really on the move, Willie learns how to walk in this part of **CLIFFHANGER***

❑ *If you find that writing music is a chore, stave off the blues with a noteworthy **MUSIC COMPOSER** program*

❑ *Find out about the techniques for **SAVING INFORMATION** from a program*

❑ *See if you can outguess the computer when you try to **MATCH THAT***





ASK YOUR NEWSAGENT FOR INPUT